

Built in objects in JavaScript


1. Arrays
2. Functions
3. Objects (Math object)
4. Built-in Methods
5. Events

Introduction

- To solve different kinds of problems, JavaScript provides various **built-in objects**.
 - Each object contains **properties and methods**.
 - Some of the built-in objects in JavaScript are:
 - Array
 - Date
 - Math
 - String
 - Number

What are built-in methods?

- A JavaScript **method** is a property containing a **function definition**. In other words, when the data stored on an object is a function we call that a method.
- Differences between properties and methods:
 - *A property is what an object has, while a method is what an object does.*

<u>Object</u>	<u>Properties</u>	<u>Methods</u>
	car.name = Fiat car.model = 500 car.weight = 850kg car.color = white	car.start() car.drive() car.brake() car.stop()

Part 1

Array in JavaScript

Array

- An **indexed list** of elements
 - Arrays are groups of numbered variables. Each variable within the array has a unique index, and is called an element of the array.
 - Elements of an array can be of any type;
 - you can have an array containing other arrays

USES OF ARRAY

- To create and store a list of elements of similar values in some specific order for access later on.

Array Index and Identifier

- Just like C, C++ and Java, the **first element** of an array has an index number equal to **zero (0)**
- Example: There are many ways of assigning identifiers to the following fruit



strawberry
fruit1
fruit[0]



orange
fruit2
fruit[1]



apple
fruit3
fruit[2]



watermelon
fruit4
fruit[3]

Array Object

- In JavaScript, arrays are implemented in the form of the 'Array' object
- The key property of the 'Array' object is 'length', i.e the number of elements in an array. E.g.

```
fruit = new Array( 4 )
```

- fruit[0], fruit[1], fruit[2], and fruit[3] are the elements of an array
- 'fruit' is the identifier for that array
- The length of the 'fruit' array is 4, i.e. 'fruit' has four elements

JavaScript Arrays are Heterogeneous

Unlike many other popular languages, a JavaScript Array can hold elements of multiple data types, simultaneously

In JavaScript, Python, and Ruby, you can have an array that stores elements of different types.

```
a = new Array( 9 ) ;
```

```
b = new Array( 13 ) ;
```

```
b[ 0 ] = 23.7 ;
```

```
b[ 1 ] = "Bhola Continental Hotel" ;
```

```
b[ 2 ] = a ;
```


Array Versus Variable

- a **variable** is a container that holds **a** value.
- an **Array** is a container that holds **multiple** values.
- Unlike variables, arrays must be declared.
- The **naming rules** for **Array Identifiers** = **Variable Identifiers**

Arrays Versus Variable (C0nt.)

Variables

```
let car1, car2, car3;
```

OR

```
let car1 = "Saab";  
let car2 = "Volvo";  
let car3 = "BMW";
```

```
document.write( car1 );  
document.write( car2 );  
document.write( car3 );
```

Arrays

```
const cars =["Saab", "Volvo", "BMW"];  
//array declaration
```

OR

```
car[ 0 ] = "Saab" ;  
car[ 1 ] = "Volvo" ;  
car[ 2 ] = "BMW" ;
```

```
for ( x = 0 ; x < 4 ; x = x + 1 ) {  
    document.write( car[ x ] ) ;  
}
```

Array Literals

- A **literal** is a value such as a literal string or a number that are comma separated.
- JavaScript has array *literals*, written with brackets and commas
 - Example: `color = ["red", "yellow", "green", "blue"];`
 - Arrays are *zero-based*: `color[0]` is "red"
- If you put two commas in a row, the array has an “empty” element in that location
 - Example: `color = ["red", , , "green", "blue"];`
 - `color` has 5 elements
- A single comma at the end is ignored
 - Example: `color = ["red", , , "green", "blue",,];` still has 5 elements

How to Create an Array

1. Use an **array literal**:

```
const array_name = [item1, item2, ...];
```

Or

```
const cars = [  
  "Saab",  
  "Volvo",  
  "BMW"  
];
```

You can also create an empty array, and then provide the elements:

```
const cars = [ ];  
cars[0]= "Saab";  
cars[1]= "Volvo";  
cars[2]= "BMW";
```

2. Use the **new** keyword.

a) You can use **new Array()**

```
const cars = new Array("Saab", "Volvo", "BMW");
```

How to Create an Array

1. Use the **new** keyword - **JavaScript new Array()** (Continue)

JavaScript has a built-in array constructor `new Array()`.

But you can safely use `[]` instead.

Example 1:

These two different statements both create a new empty array named `points`:

```
const points = new Array();  
const points = [];
```

Example 2:

These two different statements both create a new array containing 6 numbers:

```
const points = new Array(40, 100, 1, 5, 25, 10);  
const points = [40, 100, 1, 5, 25, 10];
```

Assigning Values to Arrays

- After the array is set up, you can refer to any element just like a variable.

Example,

```
subtotals = new Array(10);
```

```
subtotals[0] = 5;
```

```
subtotals[1] = 2;
```

```
subtotals[2] = a;
```

```
•
```

```
•
```

Parallel (Corresponding) Arrays

- Parallel arrays gives you the ability to store multiple pieces of information about a single entity in an application.
- The indexes of each array should correspond to one another for individual entities: Example
 - Student ID Student Name Student Exam Score

Conceptual Example of Parallel Arrays

Array Name = strSID

0	047254
1	038546
2	024136
3	057866
4	015543
5	025769
6	025119
7	035176
8	036585
9	028116

Array Name = strn201Class

0	John
1	Mary
2	Elizabeth
3	Omar
4	Charles
5	Ravi
6	Katherine
7	Hannah
8	Alexander
9	William

Array Name = fltExam1Scores

0	93
1	86
2	74
3	92
4	56
5	82
6	89
7	82
8	75
9	77

Creating Parallel Arrays

- To create parallel arrays, just create them as you would any other array:



```
let SID = new Array();  
let n201Class = new Array();  
let examScr = new Array();
```

- The arrays are separate arrays in memory, but your program should treat them as one ...

Assigning Values to Parallel Arrays

- To assign values for a single entity to parallel arrays, use the same index number for each assignment:

```
SID[5] = "025769";
```

```
n201Class[5] = "Ravi";
```

```
examScr[5] = 82;
```

Accessing Array Elements

- By referring to the index number:

```
const person = [];  
person[0] = "John";  
person[1] = "Doe";  
person[2] = "Peter";  
person.length; // Will return 3  
person[0]; // Will return "John"
```

the **length** property, will give you a count of how many indexes the array has.

- Access the Last Array Element

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];  
let fruit = fruits[fruits.length - 1];
```

Accessing Array Elements

- **Access the Full Array**

By referring to the array name:

```
const cars = ["Saab", "Volvo", "BMW"];  
document.getElementById("demo").innerHTML = cars;
```

Looping Array Elements

- Using a **for** loop:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];  
let fLen = fruits.length;  
let text = "<ul>";  
for (let i = 0; i < fLen; i++) {  
    text += "<li>" + fruits[i] + "</li>";  
}  
text += "</ul>";
```

- Using **Array.forEach()** function:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];  
let text = "<ul>";  
fruits.forEach(myFunction);  
text += "</ul>";  
function myFunction(value) {  
    text += "<li>" + value + "</li>";  
}
```

Adding Array Elements

- Using a **push()** method:

```
const fruits = ["Banana", "Orange", "Apple"];  
fruits.push("Lemon");
```

- Using **length** property:

```
const fruits = ["Banana", "Orange", "Apple"];  
fruits[fruits.length] = "Lemon";
```

Changing an Array Element

```
const cars = ["Saab", "Volvo", "BMW"];  
cars[0] = "Opel";
```

Converting an Array to a String

The JavaScript method `toString()` converts an array to a string of (comma separated) array values.

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML =  
fruits.toString();
```

Array Methods

Method	Description
<u>concat()</u>	Joins two or more arrays, and returns a copy of the joined arrays
<u>copyWithin()</u>	Copies array elements within the array, to and from specified positions
<u>entries()</u>	Returns a key/value pair Array Iteration Object
<u>every()</u>	Checks if every element in an array pass a test
<u>fill()</u>	Fill the elements in an array with a static value
<u>filter()</u>	Creates a new array with every element in an array that pass a test
<u>find()</u>	Returns the value of the first element in an array that pass a test
<u>findIndex()</u>	Returns the index of the first element in an array that pass a test
<u>forEach()</u>	Calls a function for each array element
<u>from()</u>	Creates an array from an object
<u>includes()</u>	Check if an array contains the specified element
<u>indexOf()</u>	Search the array for an element and returns its position
<u>isArray()</u>	Checks whether an object is an array
<u>join()</u>	Joins all elements of an array into a string
<u>keys()</u>	Returns a Array Iteration Object, containing the keys of the original array
<u>lastIndexOf()</u>	Search the array for an element, starting at the end, and returns its position
<u>unshift()</u>	Adds new elements to the beginning of an array, and returns the new length
<u>valueOf()</u>	Returns the primitive value of an array

Array Methods (Cont.)

Method	Description
<u>map()</u>	Creates a new array with the result of calling a function for each array element
<u>pop()</u>	Removes the last element of an array, and returns that element
<u>push()</u>	Adds new elements to the end of an array, and returns the new length
<u>reduce()</u>	Reduce the values of an array to a single value (going left-to-right)
<u>reduceRight()</u>	Reduce the values of an array to a single value (going right-to-left)
<u>reverse()</u>	Reverses the order of the elements in an array
<u>shift()</u>	Removes the first element of an array, and returns that element
<u>slice()</u>	Selects a part of an array, and returns the new array
<u>some()</u>	Checks if any of the elements in an array pass a test
<u>sort()</u>	Sorts the elements of an array
<u>splice()</u>	Adds/Removes elements from an array
<u>toString()</u>	Converts an array to a string, and returns the result
<u>pop()</u>	Removes the last element of an array, and returns that element
<u>push()</u>	Adds new elements to the end of an array, and returns the new length
<u>reduce()</u>	Reduce the values of an array to a single value (going left-to-right)
<u>reduceRight()</u>	Reduce the values of an array to a single value (going right-to-left)
<u>reverse()</u>	Reverses the order of the elements in an array

Array Methods (Continue)

- If **myArray** is an array,
 - **myArray.sort()** sorts the array alphabetically
 - **myArray.sort(function(a, b) { return a - b; })** sorts numerically in **ascending** order
 - **myArray.reverse()** reverses the array elements
 - **myArray.push(...)** adds any number of new elements to the end of the array, and increases the array's length
 - **myArray.pop()** removes and returns the last element of the array, and decrements the array's length
 - **myArray.toString()** returns a string containing the values of the array elements, separated by commas

Array Methods: `sort()`

Sorts the elements in alphabetical order

```
x = new Array ( 4 ) ;
```

```
x[ 0 ] = "Waseem" ;
```

```
x[ 1 ] = "Waqar" ;
```

```
x[ 2 ] = "Saqlain" ;
```

```
x[ 3 ] = "Shoaib" ;
```

```
x.sort() ;
```

```
for ( k = 0 ; k < x.length; k = k + 1 ) {
```

```
document.write( x[ k ] + "<br />" ) ;
```

Saqlain
Shoaib
Waqar
Waseem

Sorting Elements In Ascending Or Descending Order

- Sort numbers in an array in ascending order:
`var points = [40, 100, 1, 5, 25, 10];`
`points.sort(function(a, b){return a-b});`
- Sort numbers in an array in descending order:
`var points = [40, 100, 1, 5, 25, 10];`
`points.sort(function(a, b){return b-a});`

Array Methods: `reverse()` & `sort()`

Reverses then sort elements and also sort them

```
x = new Array ( 4 ) ;
```

```
x[ 0 ] = "Waseem" ;
```

```
x[ 1 ] = "Waqar" ;
```

```
x[ 2 ] = "Saqlain" ;
```

```
x[ 3 ] = "Shoaib" ;
```

```
x.reverse( ) ;
```

```
x.sort( ) ;
```

```
for ( k = 0 ; k < x.length; k = k + 1 ) {
```

```
document.write( x[ k ] + "<br />" ) ;
```

Saqlain
Shoaib
Waqar
Waseem

Array Methods: `sort()` & `reverse()`

sort and Reverses the order of the elements

```
x = new Array ( 4 ) ;
```

```
x[ 0 ] = "Waseem" ;
```

```
x[ 1 ] = "Waqar" ;
```

```
x[ 2 ] = "Saqlain" ;
```

```
x[ 3 ] = "Shoaib" ;
```

```
x.sort( ) ;
```

```
x.reverse( ) ;
```

```
for ( k = 0 ; k < x.length; k = k + 1 ) {
```

```
document.write( x[ k ] + "<br />" ) ;
```

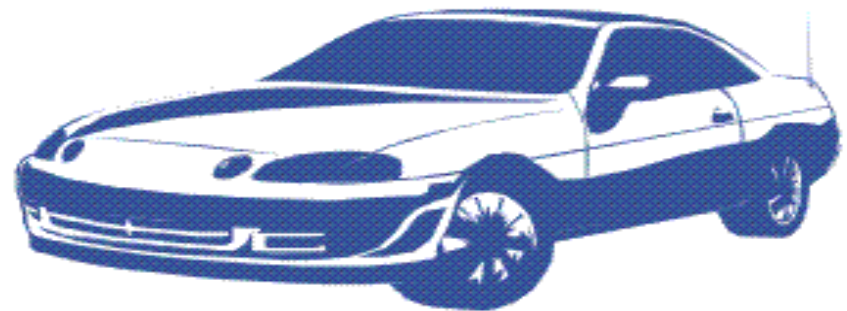
Waseem
Waqar
Shoaib
Saqlain

Part 2

Objects in JavaScript

Objects

- Objects in JavaScript are a collection of properties, each of which can contain a value.
- Each value stored in the properties can be a value, another object, or even a function.
- You can define your own objects with JavaScript, or use the several built-in objects.

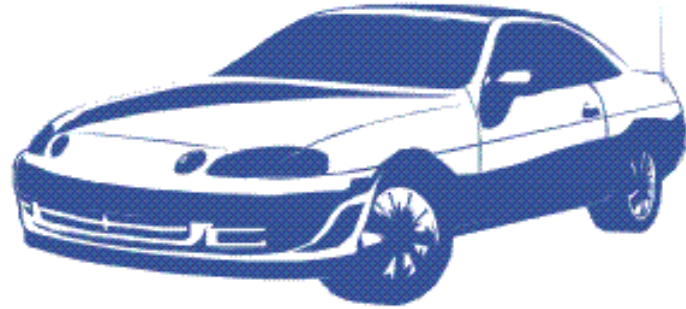


'Instances' of an Object



***All instances
of an object are objects themselves!***

'Property' Values of the Instances May Differ



Three Ways To Create An Object

- Objects are created with curly braces,

Syntax: `var myObject = { };`

1. You can use an object literal:

```
var course = { "Course_Code": " INF313 ", teacher: " Dr. Halima " }  
var car = { "identifier": "1", "name": "vw saloon" }
```

2. You can use **new** to create a “blank” object, and add fields to it later:

```
var course = new Object();  
course.number = "INF313";  
course.teacher = "Dr. Halima";
```

3. You can write and use a constructor:

```
function Course(n, t) { // best placed in <head>  
    this.number = n;    // keyword "this" is required, not optional  
    this.teacher = t;}  
var course = new Course("INF313", " Dr. Halima ");
```

Object literals

- A JavaScript object literal is a comma-separated list of name-value pairs wrapped in curly braces.
- JavaScript has object *literals*, written with this syntax:
`{ name1 : value1 , ... , nameN : valueN }`
- Object literal **property values can be of any data type**, including array literals, functions, and nested object literals.

Object Literal Syntax

- A colon separates property name[1] from value.
- A comma separates each name-value pair from the next.
- A comma after the last name-value pair is optional.

Example:

```
var myObject = {  
    sProp: 'some string value',  
    numProp: 2,  
    bProp: false  
};
```

Arrays and objects

- Arrays *are* objects

```
car = { myCar: "Saturn", 7: "Mazda" }
```

- `car[7]` is the same as `car.7`
- `car.myCar` is the same as `car["myCar"]`
- If you *know* the name of a property, you can use dot notation: `car.myCar`
- If you *don't know* the name of a property, but you have it in a variable (or can compute it), you *must* use array notation:
`car["my" + "Car"]`

The length of an array

- If **myArray** is an array, its length is given by **myArray.length**
- Array length can be changed by assignment beyond the current length
 - Example:
var myArray = new Array(5);
myArray[10] = 3;
- Arrays are **sparse**, that is, space is only allocated for elements that have been assigned a value
 - Example: **myArray[50000] = 3;** is perfectly OK
 - But indices must be between 0 and $2^{32}-1$
- As in C and Java, there are no two-dimensional arrays; but you can have an array of arrays: **myArray[5][3]**

The 'length' Property of Arrays

'd' is an instance of the 'Array' object

'length' is a property of the object 'd'

```
d = new Array ( 5 ) ;  
document.write( d.length ) ;
```

Part 3

Functions

Functions

- Functions are **groups of JavaScript statements** that have been combined under a single name. The true power of functions is to execute several statements at once.
 - Several built-in functions, such as **document.write();**
 - You can also define your own functions.
- parameters are passed *by value*,
- Objects are passed *by reference*
- defined in the **<head>** of an HTML page, to ensure that they are loaded first.

Defining and Calling Functions (Cont.)

- Use the **function** keyword.

- The syntax:

function *name*(*arg1*, ..., *argN*) { *statements* }

- The function may contain **return** *value*; statements
- Any variables declared within the function are local to it

- The syntax for calling a function is,

name(*arg1*, ..., *argN*)

Example:

```
function Linebreak( ) {  
  document.write("<br />");  
}
```

example:

```
Linebreak();
```

Defining and Calling Functions (Cont.)

```
<html>
<head>
<script language="javascript">
function calculate()
{ var X = 5;
  var Y = 4;
  var Z= X*Y;
  var xval="X has the value: ";
  var yval="Y has the value: ";

document.write(" "+xval+" "+X+" "+", "+yval+" "+Y+" "+", "+X*Y+" is
equal "+Z+"");
}
</script>
</head>
<body>
<script language="javascript">
calculate();
</script>
</body>
</html>
```

Parsing Functions

- `parseInt(string, radix)` -- returns the first integer in the string.
 - The **radix** argument specifies the base in which the number is represented in the string, e.g., 16 (hexadecimal), 10 (decimal), or 2 (binary).
- Example:

```
parseInt("313 Gilbreath", 10);
```

would return **313**

- If the first character is not a number, then the function returns "**NaN**" indicating the value is not a number.

Parsing Functions (continued)

- *parseFloat(string)* – returns the first floating point number in the string.

- Example:

```
parseFloat("2.98% of students");
```

would return 2.98

- If the first character is not a number, then the function returns "NaN" indicating the value is not a number. This includes characters such as \$ or #.

isNaN()

- `isNaN(value)` – returns a true or false based on whether value represents a number or not.
- "value" can be a string containing a number.
- Helpful with validation of forms.
- Examples:
 - `isNaN("David Tarnoff")` would return true
 - `isNaN(4*5)` would return false
 - `isNaN("315")` would return false

unescape()

- **unescape(encodedstring)**
 - goes through a string replacing escape characters with original characters.
- In some cases, strings are encountered that have certain characters replaced with escape characters.
 - example, a **URL** often replaces spaces with **%20** and the **'@'** symbol with **%40**

Example:

```
document.write(  
    unescape(" My%20e-  
mail%20is%3A%20tarnoff%  
40ucc.edu%21"));
```

would output as:

My e-mail is: tarnoff@ucc.edu!

Regular expressions

- A regular expression can be written in either of two ways:
 - Within slashes, such as **re = /ab+c/**
 - With a constructor, such as **re = new RegExp("ab+c")**
- **string.match(regex)** searches *string* for an occurrence of *regex*
 - It returns **null** if nothing is found
 - If *regex* has the **g** (global search) flag set, **match** returns an array of matched substrings
 - If **g** is not set, **match** returns an array whose 0th element is the matched text, extra elements are the parenthesized subexpressions, and the **index** property is the start position of the matched substring

Example1: Using built-in functions, variables

```
<html>
<head>
<title> JavaScript 1 </title>
</head>
<body>
<script language="JavaScript">
document.write("<h1>The date is. </h1>");
  document.write(Date());
</script>
```

```
This is the rest of the page.
</body>
```

```
</html>
```

Example2: Using built-in functions, variables

```
<html> <head> <title> JavaScript 1 </title>
```

```
<script language="JavaScript">
```

```
<!--
```

```
var rawDate = Date();
```

**Variable set to what
Date() returns.**

```
var mon = rawDate.substr(4,3);
```

**Extract the
month.**

```
document.write("The month is ");
```

```
document.write(mon);
```

```
//-->
```

```
</script> </head>
```

```
<body>
```

```
<br>
```

This is the rest of the page.

```
</body> </html>
```

Example3: Form Validation

```
<html>
<head><title>Form example </title>
<script language="JavaScript">
function verify(f)
{

if (f.lname.value == null || f.address.value == null)
{
    alert("Form needs a last name and an address");
    return false;
}
if (f.lname.value == "" || f.address.value == "")
{
    alert("Form needs a last name and an address");
    return false;
}
return true;

}
</script>
</head>
```

Example3: Form Validation (Continue)

```
<body>
<h1> Address Information </h1> <br>
<form method=post enctype="text/plain" action=""
onSubmit="return verify(this);">
```

JavaScript
Event

```
First Name: <input type="text" name="fname"> <br />
Last Name: <input type="text" name="lname"> <br />
Street Address: <input type="text" name="address" size=30> <br />
Town/City: <input type="text" name="city"> <br />
State: <select name="state" size=1> <br />
<option value="NY" selected> Utah
<option value="NY" selected> Idaho
<option value="NY" selected> New York
<option value="NJ"> New Jersey
<option value="CT"> Connecticut
<option value="PA"> Pennsylvania
</select> <br />
Status: <input type="radio" name="status" value="R"> Returning client
<input type="radio" name="status" value="N"> New client
<hr /> Thank you <p>
<input type="submit" value="Send information">
</form> </body> </html>
```

Part 4

Events

JavaScript Events

- Events are things that happen to an object or in the browser
- Events are actions that can be detected by JavaScript.
- Every element on a web page has certain events which can trigger JavaScript functions.
- Often placed within the HTML tag
 - `<tag attribute1 attribute2 onEventName="javascript code;">`
 - ``

JavaScript Events (Cont.)

- Example, assume we have an object "**person**".
- An event might be that their eyes become dry. What would they do? Blink!

```
person.onDryEyes = blink();
```

- The object in this example is "**person**".
- method or function is "**blink()**".
- The **event** is "**onDryEyes**".

JavaScript Events

- At times, you won't even need to use the `<script>` tag to include JavaScript in a document.
- Instead, you can use **event handlers** (These are special attributes for HTML tags, and can be used to respond to events). E.g.
- Examples of **event handlers** for your HTML pages include:
 1. `onload`
 2. `onClick`
 3. `onMouseOver`
 4. `onMouseOut`
- Each of these events handlers can execute a script.
- Events handlers must be placed in a tag.

Example:

```
<a href="somesite.htm" onClick =  
"javascript:function();" >link text</a>
```

```
<a href="next.html"  
onMouseOver="alert( 'hello!' );" >link text</a>
```


Common JavaScript Event Handlers

Event	Occurs when...	Event Handler
• click	User clicks on form element or link	onClick
• change	User changes value of text, textarea, or select element	onChange
• focus	User gives form element input focus	onFocus
• blur	User removes input focus from form element	onBlur
• mouseover	User moves mouse pointer over a link or anchor	onMouseOver
• mouseout	User moves mouse pointer off of link or anchor	onMouseOut
• select	User selects form element's input field	onSelect
• submit	User submits a form	onSubmit
• resize	User resizes the browser window	onResize
• load	User loads the page in the Navigator	onLoad
• unload	User exits the page	onUnload

The Document Object Model

- The set of all events which may occur and the page elements on which they can occur is part of the Document Object Model(DOM) not Javascript
 - Browsers don't necessarily share the same set of events
- When a document is loaded in the web browser, a number of objects are created. Most commonly used are **window** and **document**
- **Window**
 - **open(), close(), alert(), confirm(), prompt()**
- **Document**
 - Contains arrays which store all the components of your page
 - You can access and call methods on the components using the arrays
 - An object may also be accessed by its name
 - **document.myform.address.value = "123 Main"**
 - **document.myform.reset()**
 - Can also search for element by name or id
 - **document.getElementById("myelementid")**
 - **document.getElementsByName("myelementname")**

In Class Exercises

Study the code below and correct it:

```
const person = [];  
person["firstName"] = "John";  
person["lastName"] = "Doe";  
person["age"] = 46;  
person.length;  
person[0];
```

LAB Exercise

Lab 1

1. Write JavaScript code to do the following: collect two numbers (using prompt) and display the larger.

```
<script language="javascript" type="text/javascript">  
<!--  
var n1 = 1*prompt("Please enter a number.");  
var n2 = 1*prompt("Please enter another number.");  
if (n1 > n2)  
alert(n1+" is the larger.");  
else  
alert(n2+" is the larger.");  
//-->  
</script>
```

Lab 2

2. Write JavaScript code to do the following: Read two numbers (using prompt) and display their total/sum.

ANSWER:

```
<script language="javascript" type="text/javascript">  
<!--  
var n1 = prompt("Please enter a number.");  
var n2 = prompt("Please enter another number.");  
alert("The sum of "+n1+" and "+n2+" is "+  
parseInt(n1)+ parseInt (n2);  
//-->  
</script>
```

Lab 3

1. Write JavaScript code to do the following: Read two numbers (using prompt) and display their modulo.

ANSWER:

```
<script language="javascript" type="text/javascript">  
<!--  
var n1 = prompt("Please enter a number.");  
var n2 = prompt("Please enter another number.");  
alert("The modulo of "+n1+" and "+n2+" is "+ parseInt(n1)  
% parseInt (n2);  
//-->  
</script>
```

Lab 4

- Write a function to do the following.; display the product of $0 * 1 * 2 * \dots * n$.

```
function product(n)
{
var a = 0;
for (var i=0;i<=n;i++)
a=a*i;
alert("The Product of 0 through "*n*" is "+a);
}
```

Prepared by: Dr. Alimatu - Saadia Yussiff

ASSIGNMENT

Assignment 3

Due Date: 27/08/2023

Develop a Web page that **prompts** the user for **10 words**, and then displays them in the form of a list in two different ways:

1. In the order in which the words were entered
2. In a sorted order

Explorer User Prompt

Script Prompt:

Enter word # 0

OK

Cancel

Unsorted Words:

Waseem

Waqar

Suzuki

Apple

Mac

Dragon

Saqlain

Sajid

Book

center

Sorted Words:

Apple

Book

Dragon

Mac

Sajid

Saqlain

Suzuki

Waqar

Waseem

center

Prepared by: Dr. Alimatu - Saadia Yussiff

Pseudo Code

1. **Declare the array** that will be used for storing the words
2. **Prompt** the user and **read** the user input into the elements of the array
3. Now **write/display** the array to the document
4. **Sort** the array
5. **Write/display** the sorted array to the document

```
<HTML>
  <HEAD>
    <TITLE>Sort Ten Words</TITLE>
    <SCRIPT>
      words = new Array ( 10 ) ;
      for ( k = 0 ; k < words.length ; k = k + 1 ) {
        words[ k ] = window.prompt( "Enter word # " + k, "" ) ;
      }
      document.write( "UNSORTED WORDS:" + "<BR>" ) ;
      for ( k = 0 ; k < words.length ; k = k + 1 ) {
        document.write( words[ k ] + "<BR>" ) ;
      }
      words.sort( ) ;
      document.write( "SORTED WORDS:" + "<BR>" ) ;
      for ( k = 0 ; k < words.length ; k = k + 1 ) {
        document.write( words[ k ] + "<BR>" ) ;
      }
    </SCRIPT>
  </HEAD>
  <BODY>
    </BODY>
</HTML>
```

Pseudo Code

1. **Declare the array** that will be used for storing the words
2. **Prompt** the user and **read** the user input into the elements of the array
3. Now write the array to the document
4. Sort the array
5. Write the sorted array to the document

```
words = new Array ( 10 ) ;  
  
for ( k = 0 ; k < words.length ; k = k + 1 ) {  
    words[ k ] = window.prompt(  
        "Enter word # " + k, "" ) ;  
}
```

This method is used for collecting data from the user. It can display a message and provides a field in which the user can enter data

Pseudo Code

1. Declare the array that will be used for storing the words
2. Prompt the user and read the user input into the elements of the array
3. Now **write** the array to the document
4. Sort the array
5. Write the sorted array to the document

```
document.write( "Unsorted Words:" + "<BR>" );
```

```
for ( k = 0 ; k < words.length ; k = k + 1 ) {           document.write(  
words[ k ] + "<BR>" );  
}
```

Pseudo Code

1. Declare the array that will be used for storing the words
2. Prompt the user and read the user input into the elements of the array
3. Now write the array to the document
4. **Sort** the array
5. **Write** the sorted array to the document

```
words.sort( ) ;
```

```
document.write( "Sorted Words:" + "<BR>" ) ;
```

```
for ( k = 0 ; k < words.length ; k = k + 1 ) { document.write(  
words[ k ] + "<BR>" ) ;  
}
```

Exercise

Build a Web page that implements the **Bubble Sort** algorithm
The numbers to be sorted will be created by you and should be **hard coded** in the JavaScript code.

Your page should display a **button** labeled “**Display Numbers**”.
When that button is clicked, the page should display the unsorted list of numbers

Your page should display a second **button** labeled “**Run Bubble Sort**”. When that button is clicked, the page should display the sorted list of numbers