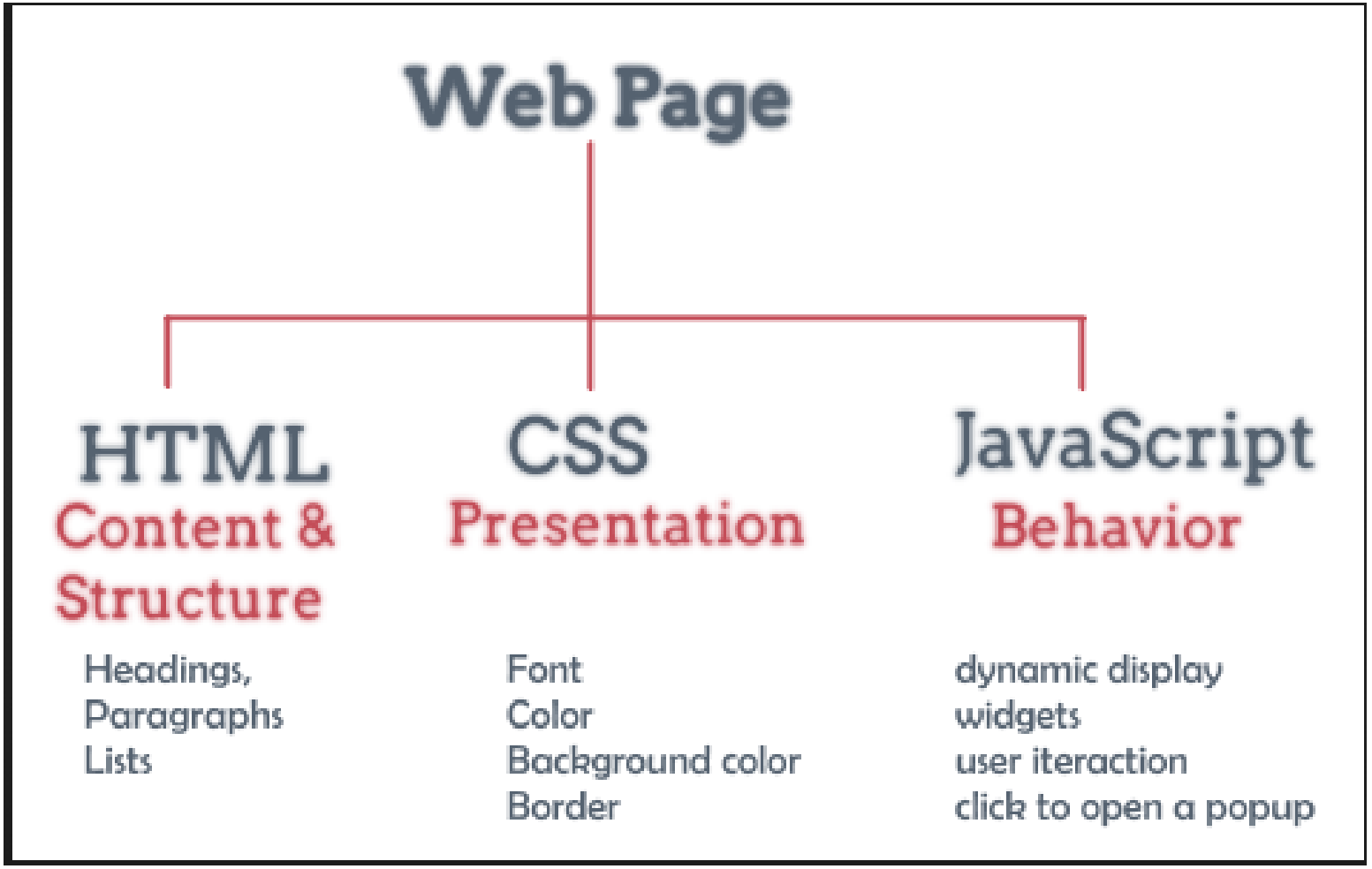
INTRODUCTION TO JAVASCRIPT

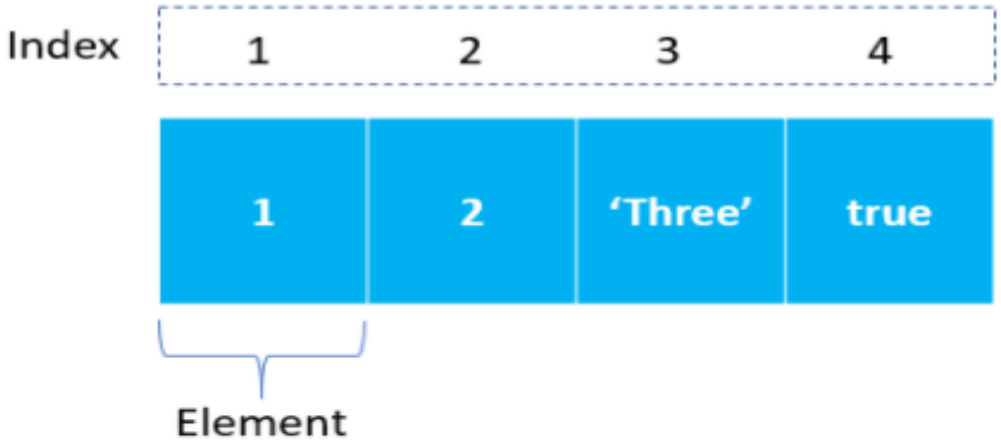
Presented by
Eng./Abanoub Nabil
Teaching assistant at ITI

JavaScript …


Web Page
HTML
Content & Structure
Headings,
Paragraphs
Lists
CSS
Presentation
Font
Color
Background color
Border
JavaScript
Behavior
dynamic display
widgets
user iteraction
click to open a popup

Introduction to JavaScript arrays

- In JavaScript, an array is an ordered **list of values**. Each value is called an element specified by an **index**.



An JavaScript array has the following characteristics:

1. First, an array can hold values of different types. For example, you can have an array that stores the number and string, and boolean values.
2. Second, the length of an array is dynamically sized and auto-growing. In other words, you don't need to specify the array size upfront.

Introduction to JavaScript arrays

- **Creating JavaScript arrays**
- JavaScript provides you with **two ways to create an array**.

  1-The first one is to use the Array constructor as follows:

```
var grades = new Array();
```

- The **grades** array is **empty** i.e. it holds no element.
- **If you know** the number of elements that the array will hold, you can create an array with an initial size as shown in the following example:

```
var grades = new Array(10);
```

Introduction to JavaScript arrays

- To create an array with **some elements**, you pass the elements as a comma-separated list into the Array() constructor.
- For example, the following creates the grades array that has five elements (or numbers):

```
var grades = new Array(9,10,8,7,6);
```

- JavaScript allows you **to omit the new operator** when you use the array constructor. For example, the following statement creates the grades array.

```
var grades =  Array(10);
```

Introduction to JavaScript arrays

- The second way to create an array is to use the **array literal notation**:

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

- **Example**

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

Introduction to JavaScript arrays

- **Accessing JavaScript array elements**
- JavaScript arrays are zero-based indexed. In other words, the first element of an array starts at index 0, the second element starts at index 1, and so on.
- To access an element in an array, you specify an index in the square brackets []:

```
arrayName[index]
```

```
var names=['Ahmed','Hany','Abanoub','Christeen']

names[0] //Ahmed
names[3] //Christeen
names[7] //undefined
```

Introduction to JavaScript arrays

- **Array Properties and Methods**
- The real strength of JavaScript arrays are the built-in array properties and methods:
- Examples

```
var x = cars.length;   // The length property returns the number of elements
var y = cars.sort();   // The sort() method sorts arrays
```

- **The length Property**
- The length property of an array returns the length of an array (the number of array elements).

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.length;   // the length of fruits is 4
```

Introduction to JavaScript arrays

- **Looping Array Elements**

The safest way to loop through an array, is using a `for` loop:

Example

```
var fruits, text, fLen, i;
fruits = ["Banana", "Orange", "Apple", "Mango"];
fLen = fruits.length;

text = "<ul>";
for (i = 0; i < fLen; i++) {
  text += "<li>" + fruits[i] + "</li>";
}
text += "</ul>";
```

Introduction to JavaScript arrays

- **Adding Array Elements**

The easiest way to add a new element to an array is using the `push()` method:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Lemon");    // adds a new element (Lemon) to fruits
```

New element can also be added to an array using the `length` property:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Lemon";    // adds a new element (Lemon) to fruits
```

Introduction to JavaScript arrays

- A common question is: How do I know if a variable is an array?
- The problem is that the JavaScript operator typeof returns "object":

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];

typeof fruits;    // returns object
```

- **Solution 1:**

To solve this problem ECMAScript 5 defines a new method `Array.isArray()`:

```
Array.isArray(fruits);   // returns true
```

Introduction to JavaScript arrays

- **Solution 2**

The `instanceof` operator returns true if an object is created by a given constructor:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits instanceof Array;   // returns true
```

Introduction to JavaScript arrays

- **JavaScript Array Methods (toString)**

Converting Arrays to Strings

The JavaScript method `toString()` converts an array to a string of (comma sepa

Example

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

Introduction to JavaScript arrays

- **JavaScript Array Methods (Join)**

The `join()` method also joins all array elements into a string.

It behaves just like `toString()`, but in addition you can specify the separator:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
```

Result:

Banana * Orange * Apple * Mango

Introduction to JavaScript arrays

- **JavaScript Array Methods (pop)**

Popping

The `pop()` method removes the last element from an array:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();                 // Removes the last element ("Mango") from fruits
```

Introduction to JavaScript arrays

- **JavaScript Array Methods (push)**

The `push()` method adds a new element to an array (at the end):

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi");        //  Adds a new element ("Kiwi") to fruits
```

The `push()` method returns the new array length:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var x = fruits.push("Kiwi");    //  the value of x is 5
```

Introduction to JavaScript arrays

- **JavaScript Array Methods (shift)**

Shifting Elements

Shifting is equivalent to popping, working on the first element instead of the last.

The `shift()` method removes the first array element and "shifts" all other elements to a lower index.

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.shift();            // Removes the first element "Banana" from fruits
```

Introduction to JavaScript arrays

- **JavaScript Array Methods (unshift)**

The `unshift()` method adds a new element to an array (at the beginning), and "unshifts" older elements:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");    // Adds a new element "Lemon" to fruits
```

Introduction to JavaScript arrays

- **JavaScript Array Methods (splicing)**

The `splice()` method can be used to add new items to an array:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
```

The first parameter (2) defines the position **where** new elements should be **added** (spliced in).

The second parameter (0) defines **how many** elements should be **removed**.

The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be **added**.

The `splice()` method returns an array with the deleted items:

Introduction to JavaScript arrays

- **JavaScript Array Methods (splicing)**

Using splice() to Remove Elements

With clever parameter setting, you can use `splice()` to remove elements without leaving "holes" in the array:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(0, 1);         // Removes the first element of fruits
```

Introduction to JavaScript arrays

- **JavaScript Array Methods (concat)**

The `concat()` method creates a new array by merging (concatenating) existing arrays:

Example (Merging Two Arrays)

```
var myGirls = ["Cecilie", "Lone"];
var myBoys = ["Emil", "Tobias", "Linus"];
var myChildren = myGirls.concat(myBoys);   // Concatenates (joins) myGirls and myBoys
```

The `concat()` method can take any number of array arguments:

Example (Merging Three Arrays)

```
var arr1 = ["Cecilie", "Lone"];
var arr2 = ["Emil", "Tobias", "Linus"];
var arr3 = ["Robin", "Morgan"];
var myChildren = arr1.concat(arr2, arr3);   // Concatenates arr1 with arr2 and arr3
```

Introduction to JavaScript arrays

- **JavaScript Array Methods (slice)**

The `slice()` method slices out a piece of an array into a new array.

This example slices out a part of an array starting from array element 1 ("Orange"):

Example

```
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(1);
```

Introduction to JavaScript arrays

- **JavaScript Array Methods (slice)**

The `slice()` method can take two arguments like `slice(1, 3)`.

The method then selects elements from the start argument, and up to (but not including) the end argument.

Example

```
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(1, 3);
```

JavaScript Date Objects

Creating Date Objects

Date objects are created with the `new Date()` constructor.

There are **4 ways** to create a new date object:

```
new Date()
new Date(year, month, day, hours, minutes, seconds, milliseconds)
new Date(milliseconds)
new Date(date string)
```

JavaScript Date Objects

new Date()

`new Date()` creates a new date object with the **current date and time**:

Example

```
var d = new Date();
```

new Date(*year, month, ...*)

`new Date(year, month, ...)` creates a new date object with a **specified date and time**.

7 numbers specify year, month, day, hour, minute, second, and millisecond (in that order):

Example

```
var d = new Date(2018, 11, 24, 10, 33, 30, 0);
```

JavaScript Date Objects

6 numbers specify year, month, day, hour, minute, second:

Example

```
var d = new Date(2018, 11, 24, 10, 33, 30);
```

5 numbers specify year, month, day, hour, and minute:

Example

```
var d = new Date(2018, 11, 24, 10, 33);
```

JavaScript Date Objects

4 numbers specify year, month, day, and hour:

Example

```
var d = new Date(2018, 11, 24, 10);
```

3 numbers specify year, month, and day:

Example

```
var d = new Date(2018, 11, 24);
```

JavaScript Date Objects

2 numbers specify year and month:

Example

```
var d = new Date(2018, 11);
```

You cannot omit month. If you supply only one parameter it will be treated as milliseconds.

Example

```
var d = new Date(2018);
```

JavaScript Date Objects

new Date(*dateString*)

`new Date(dateString)` creates a new date object from a **date string**:

Example

```
var d = new Date("October 13, 2014 11:13:00");
```

JavaScript Date Objects

new Date(*milliseconds*)

`new Date(milliseconds)` creates a new date object as **zero time plus milliseconds**:

Example

```
var d = new Date(0);
```

```
var d = new Date(86400000); //plus one day 24*60*60*1000
```

JavaScript Date Objects

- **Date formats**
- More details
- https://www.w3schools.com/js/js_date_formats.asp

Type	Example
ISO Date	"2015-03-25" (The International Standard)
Short Date	"03/25/2015"
Long Date	"Mar 25 2015" or "25 Mar 2015"

JavaScript Date Objects

- ## JavaScript Get Date Methods

These methods can be used for getting information from a date object:

Method	Description
getFullYear()	Get the **year** as a four digit number (yyyy)
getMonth()	Get the **month** as a number (0-11)
getDate()	Get the **day** as a number (1-31)
getHours()	Get the **hour** (0-23)
getMinutes()	Get the **minute** (0-59)
getSeconds()	Get the **second** (0-59)
getMilliseconds()	Get the **millisecond** (0-999)
getTime()	Get the time (milliseconds since January 1, 1970)
getDay()	Get the weekday as a number (0-6)
Date.now()	Get the time. ECMAScript 5.

JavaScript Date Objects

- **JavaScript Set Date Methods**

Set Date methods are used for setting a part of a date:

Method	Description
setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

JavaScript Date Objects

- **JavaScript Set Date Methods**

3. to Methods

var myDate = new Date ("November 25,2006 11:13:00");

Name	Example	Returned Value
toUTCString()	myDate.toUTCString()	Sat, 25 Nov 2006 09:13:00 UTC
toLocaleString()	myDate.toLocaleString()	25 نوفمبر, 2006 11:13:00 ص (Based on date format in your OS)
toLocaleTimeString()	myDate.toLocaleTimeString()	11:13:00 ص
toLocaleDateString()	myDate.toLocaleDateString()	01 نوفمبر, 2006
toString()	myDate.toString()	Sat Nov 25 11:13:00 UTC+0200 2006
toDateString()	myDate.toDateString()	Sun Nov 1 2006

JavaScript Math Object

The JavaScript Math object allows you to perform mathematical tasks on numbers.

Example

```
Math.PI;            // returns 3.141592653589793
```

Math.round()

`Math.round(x)` returns the value of x rounded to its nearest integer:

Example

```
Math.round(4.7);    // returns 5
Math.round(4.4);    // returns 4
```

JavaScript Math Object

Math.pow()

`Math.pow(x, y)` returns the value of x to the power of y:

Example

```
Math.pow(8, 2);       // returns 64
```

Math.sqrt()

`Math.sqrt(x)` returns the square root of x:

Example

```
Math.sqrt(64);        // returns 8
```

JavaScript Math Object

Math.abs()

`Math.abs(x)` returns the absolute (positive) value of x:

Example

```
Math.abs(-4.7);      // returns 4.7
```

Math.ceil()

`Math.ceil(x)` returns the value of x rounded **up** to its nearest integer:

Example

```
Math.ceil(4.4);      // returns 5
```

JavaScript Math Object

Math.floor()

`Math.floor(x)` returns the value of x rounded **down** to its nearest integer:

Example

```
Math.floor(4.7);     // returns 4
```

JavaScript Math Object

Math.min() and Math.max()

`Math.min()` and `Math.max()` can be used to find the lowest or highest value in a list of arguments:

Example

```
Math.min(0, 150, 30, 20, -8, -200);  // returns -200
```

JavaScript Math Object

Math.random()

`Math.random()` returns a random number between 0 (inclusive), and 1 (exclusive):

Example

```
Math.random();      // returns a random number
```

- Complete reference
- https://www.w3schools.com/jsref/jsref_obj_math.asp

JavaScript Math Object

Math.random()

`Math.random()` returns a random number between 0 (inclusive), and 1 (exclusive):

Example

```
Math.random();              // returns a random number
```

JavaScript Random Integers

`Math.random()` used with `Math.floor()` can be used to return random integers.

Example

```
Math.floor(Math.random() * 10);     // returns a random integer from 0 to 9
```

Error handling

- **JavaScript Errors - Throw and Try to Catch**
-

The `try` statement lets you test a block of code for errors.

The `catch` statement lets you handle the error.

The `throw` statement lets you create custom errors.

The `finally` statement lets you execute code, after try and catch, regardless of the result.

Error handling

- **JavaScript Errors - Throw and Try to Catch**

Errors Will Happen!

When executing JavaScript code, different errors can occur.

Errors can be coding errors made by the programmer, errors due to wrong input, and other unforeseeable things.

Example

In this example we have written alert as adddlert to deliberately produce an error:

```
<p id="demo"></p>

<script>
try {
  adddlert("Welcome guest!");
}
catch(err) {
  document.getElementById("demo").innerHTML = err.message;
}
</script>
```

Error handling

- **JavaScript Errors - Throw and Try to Catch**
-

JavaScript try and catch

The `try` statement allows you to define a block of code to be tested for errors while it is being executed.

The `catch` statement allows you to define a block of code to be executed, if an error occurs in the try block.

The JavaScript statements `try` and `catch` come in pairs:

```
try {
  Block of code to try
}
catch(err) {
  Block of code to handle errors
}
```

Error handling

- **JavaScript Errors - Throw and Try to Catch**

JavaScript Throws Errors

When an error occurs, JavaScript will normally stop and generate an error message.

The technical term for this is: JavaScript will **throw an exception (throw an error)**.

JavaScript will actually create an **Error object** with two properties: **name** and **message**.

Error handling

- **JavaScript Errors - Throw and Try to Catch**
-

The throw Statement

The `throw` statement allows you to create a custom error.

Technically you can **throw an exception (throw an error)**.

The exception can be a JavaScript `String`, a `Number`, a `Boolean` or an `Object`:

```
throw "Too big";    // throw a text
throw 500;          // throw a number
```

Error handling

- **JavaScript Errors - Throw and Try to Catch**
-

The finally Statement

The `finally` statement lets you execute code, after try and catch, regardless of the result:

Syntax

```
try {
  Block of code to try
}
catch(err) {
  Block of code to handle errors
}
finally {
  Block of code to be executed regardless of the try / catch result
}
```

Error handling

- **onError**

```
function supError()
        {
                alert("Error occured")
        }
        window.onerror=supError;

OR
        function supError()
        {
                return true;
        }
        window.onerror=supError;
```