



CSCI 1720

JavaScript – Part 2

JavaScript Operators

JS Operators

JavaScript operators are identical to Java's

| | |
|----|----------------|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| / | Division |
| % | Modulus |
| ++ | Increment |
| -- | Decrement |

JS Assignment Operators

JavaScript operators are identical to Java's

| Operator | Example | Same as |
|----------|---------|-----------|
| = | x = y | 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 |

JS String Operators

The += assignment operator can also be used to add (concatenate) strings

```
<p>The assignment operator += can concatenate strings.</p>
<p id="demo"></p>
<script>
txt1 = "One ring to rule them all, ";
txt1 += "One ring to find them";
document.getElementById("demo").innerHTML = txt1;
</script>
```



The assignment operator += can concatenate strings.

One ring to rule them all, One ring to find them

JS Adding Strings and Numbers

Adding two numbers, will return the sum, but adding a number and a string will return a string

```
<p>Adding a number and a string, returns a string.</p>
```

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

```
<script>
var x = 5 + 5;
var y = "5" + 5;
var z = "Hello" + 5;
document.getElementById("demo").innerHTML =
x + "<br>" + y + "<br>" + z;
</script>
```



Adding a number and a string, returns a string.

10
55
Hello5

JS Comparison Operators

JavaScript operators are identical to Java's, with a couple of additional ones

| Operator | Description |
|----------|--------------------------|
| == | equal to |
| === | equal value & equal type |
| != | not equal |
| !== | not equal value or type |
| > | greater than |
| < | less than |
| >= | greater than or equal to |
| <= | less than or equal to |

JS Logical Operators

JavaScript operators are identical to Java's, with a couple of additional ones

| <u>Operator</u> | <u>Description</u> |
|-----------------|--------------------|
| && | logical and |
| | logical or |
| ! | Logical not |

JavaScript Data Types

JS Data Types

JavaScript variables can hold many data types: numbers, strings, objects and more (Remember, **var** is deprecated. Use **let** instead)

```
let length = 16;           // Number
let lastName = "Johnson"; // String
let x = { firstName:"John",lastName:"Dokes"}; // Object
```

In programming, data types is an important concept

To be able to operate on variables, it is important to know something about the type

Without data types, a computer cannot safely solve this

```
let x = 16 + "Volvo";
```

JS Data Types

JavaScript has dynamic types

This means that the same variable can be used to hold different data types

```
let x;           // x is undefined
x = 5;           // Now x is a number
x = "John";      // Now x is a string
```

JS Data Types

JavaScript has only one type of numbers

Numbers can be written with, or without decimals

```
let x1 = 34.00;    // With decimals,  
let x2 = 34;      // Without
```

Extra large or extra small numbers can be written with scientific (exponential) notation

```
let y = 123e5,    // 12300000  
    z = 123e-5;  // 0.00123
```

JS Data Types

Booleans can only have two values: true or false

```
let y = true,  
    z = false;    // Notice, no quotes
```

JS Data Types

JavaScript arrays are written with square brackets

Array items are separated by commas

The following code declares (creates) an array called cars, containing three items (car names)

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

JS Data Types

JavaScript objects are written with braces

Object properties are written as name:value pairs, separated by commas

```
<p id="demo"></p>
<script>
  let person = {
    firstName : "Joe",
    lastName  : "Dokes",
    age       : 50,
    eyeColor  : "blue"
  };

  document.getElementById("demo").innerHTML =
  person.firstName + " is " + person.age +
  " years old.";
</script>
```



Joe is 50 years old.

JS Data Types – typeof Operator

You can use the JavaScript typeof operator to find the type of a JavaScript variable

The `typeof` operator returns the type of a variable or an expression

```
typeof "";           // Returns "string"
typeof "Joe";       // Returns "string"
typeof "Joe Dokes"; // Returns "string"
typeof 0;           // Returns "number"
typeof 314;         // Returns "number"
typeof 3.14;        // Returns "number"
typeof (3);         // Returns "number"
typeof (3 + 4);     // Returns "number"
```

JS Data Types – Primitive Data

A primitive data value is a single simple data value with no additional properties and methods

The typeof operator can return one of these primitive types:

| | |
|-----------|--|
| string | <code>typeof "Joe Dokes";</code> // Returns "string" |
| number | <code>typeof 0;</code> // Returns "number" |
| Boolean | <code>typeof true;</code> // Returns "boolean" |
| null | |
| undefined | <code>typeof false;</code> // Returns "boolean" |

JS Data Types

The typeof operator can return one of two complex types:

function
object

```
typeof [1,2,3,4];           // Returns "object"  
typeof {name: "John", age: 34}; // Returns "object"  
typeof function myFunc(){   // Returns "function"
```

The typeof operator returns **object** for arrays because in JavaScript arrays are objects

JS Data Types

In JavaScript, a variable without a value, has the value undefined

The `typeof` is also undefined

```
let person; // Value is undefined, type is undef
```

JS Data Types

In JavaScript, a variable without a value, has the value undefined

The `typeof` is also undefined

```
let person; // Value is undefined, type is undef
typeof person; // Also undefined
console.log(person, typeof(person));
```

undefined 'undefined' lecture.js.html:24

An empty value has nothing to do with undefined

An empty string variable has both a value and a type

```
let car = "";
console.log(car, typeof(car));
```

string lecture.js.html:26

JS Data Types

In JavaScript null is "nothing"

It is supposed to be something that doesn't exist

Unfortunately, in JavaScript, the data type of null is an object

You can consider it a bug in JavaScript that `typeof null` is an object. It should be null

You can empty an object by setting it to null

```
var person = null; // Value is null, but type is still an object
```

You can also empty an object by setting it to undefined

JS Data Types

Difference Between Undefined and Null

```
> typeof undefined
< 'undefined'
-----
> typeof null //object
< 'object'
-----
> null === undefined
< false
-----
> null == undefined
< true
```

JavaScript Functions

JS Functions

A JavaScript function is a block of code designed to perform a particular task

A JavaScript function is executed when **something** invokes it (calls it)

```

> function multiply (p1, p2) {
    return p1 * p2;
}
< undefined
> let x = multiply(7,6);
< undefined
> console.log(x);
42
< undefined
>

> console.log(multiply(3,7));
21
< undefined
>

```

JS Function Syntax

A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ()

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables)

The parentheses may include parameter names separated by commas (parameter1, parameter2, ...)

The code to be executed, by the function, is placed inside brackets: {}

```

function name(parameter1, parameter2, parameter3) {
    code to be executed
}

```

JS Function Syntax

Function parameters are the names listed in the function definition

Function arguments are the real values received by the function when it is invoked

Inside the function, the arguments (the parameters) behave as local variables

A Function is much the same as a Procedure or a Subroutine, in other programming languages

JS Function Invocation

The code inside the function will execute when "something" invokes (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

JS Function Return

JavaScript reaches a return statement, the function will stop executing

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement

Functions often compute a return value. The return value is **returned** back to the **caller**

JS Function Example

```

<p>
  This example calls a function<br>
  which performs a calculation<br>
  and returns the result:
</p>
<p id="demo"></p>
<script>
  function multiply(a, b) {
    return a * b;
  }
  document.getElementById("demo").innerHTML =
    multiply(6, 7);
</script>

```



This example calls a function
which performs a calculation
and returns the result:

42

JS – Why Functions?

You can reuse code: Define the code once, and use it many times

You can use the same code many times with different arguments, to produce different results

Functions can be used the same way as you use variables, in all types of formulas, assignments, and calculations

```
let temp = toCelsius(77),  
    text = "Seventy-seven degrees is " + x + " Celsius";
```

JavaScript Scope

JS Scope

Scope is the set of variables you have access to

In JavaScript, objects and functions are also variables

In JavaScript, scope is the set of variables, objects, and functions you have access to

JavaScript has function scope: The scope changes inside functions

JS Scope

Variables declared within a JavaScript function, are **LOCAL** to the function

Local variables have **local scope**: They can only be accessed within the function

```
<p>
  The local variable carName cannot<br>
  be accessed from code outside the <br>
  function:
</p>

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

<script>
  printCarName();
  document.getElementById("demo").innerHTML =
  "The type of carName is " + typeof(carName);

  function printCarName() {
    let carName = "Volvo";
  }
</script>
```



The local variable carName cannot
be accessed from code outside the
function:

The type of carName is undefined

JS Scope

A variable declared outside a function, becomes GLOBAL

A global variable has global scope: All scripts and functions on a web page can access it

```

<p>
  A GLOBAL variable can be<br>
  accessed from any script or<br>
  function:
</p>

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

<script>
  let carName = "Volvo";
  printCarName();

  function printCarName() {
    document.getElementById("demo").innerHTML =
      "I can display " + carName;
  }
</script>

```



A GLOBAL variable can be
accessed from any script or
function:

I can display Volvo

JS Scope

Do **NOT** create global variables unless you intend to

Global variables, in any programming language, are
generally a bad idea

JS Scope

With JavaScript, the global scope is the complete JavaScript environment
 In HTML, the global scope is the window object; all global variables belong to it

```
<p>
  In HTML, all global variables<br>
  will become window variables.
</p>

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

<script>
  let carName = "Volvo";
  printCarName();

  function printCarName() {
    document.getElementById("demo").innerHTML =
    "I can display " + window.carName;
  }
</script>
```



In HTML, all global variables
 will become window variables.
 I can display Volvo

JS Scope

Your global variables (or functions) can overwrite window variables (or functions)

Any function, including the window object, can overwrite your global variables and functions

The lifetime of a JavaScript variable starts when it is declared

Local variables are deleted when the function is completed

In a web browser, global variables are deleted when you close the browser window (or tab), but remains available to new pages loaded into the same window

JavaScript Conditions

JS Conditions

Very often when you write code, you want to perform different actions for different decisions

You can use **conditional statements** in your code to do this

JS Conditions

In JavaScript we have the following conditional statements:

Use **if** to specify a block of code to be executed, if a specified condition is true

Use **else** to specify a block of code to be executed, if the same condition is false

Use **else if** to specify a new condition to test, if the first condition is false

Use **switch** to specify many alternative blocks of code to be executed

JS Conditions -- if

Use the if statement to specify a block of JavaScript code to be executed if a condition is true

Note that if is in lowercase letters. Uppercase letters (If or IF) will generate a JavaScript error

```
<script>
  if (condition) {
    // code to execute if true;
  }
</script>
```

JS Conditions -- if

```
<p>
  Display 'Good day' if the<br>
  hour is less that 18:00
</p>

<p id="demo">Good Evening!</p>

<script>
  if (new Date().getHours() < 18) {
    document.getElementById("demo").innerHTML =
      "Good Day!";
  }
</script>
```



Display 'Good day' if the
hour is less that 18:00

Good Day!

^ [Speaker Icon] 1:39 PM

JS Conditions – if - else

Use the else statement to specify a block of code to be executed if the condition is false

```
<script>
  if (condition) {
    // code to execute if true;
  } else {
    // code to execute if false;
  }
</script>
```

JS Conditions – if - else

```

<p>
  Click the button to display<br>
  a time-based greeting:
</p>
<button onclick="dayEveningGreet()">
  Try it
</button>

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

<script>
  function dayEveningGreet() {
    let hour = new Date().getHours(),
        greeting;
    if (hour < 18) {
      greeting = "Good day!";
    } else {
      greeting = "Good evening!";
    }
    document.getElementById("demo").
      innerHTML = greeting;
  }
</script>

```



Click the button to display
a time-based greeting:



Click the button to display
a time-based greeting:

Good day!

JS Conditions – else if

Use the else if statement to specify a new condition if the first condition is false

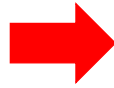
```

if (condition1) {
  // code to execute;
} else if (condition2) {
  // code to execute;
} else {
  // code to execute if both
  // condition1 and condition2
  // are false;
}

```

JS Conditions – else if

```
<script>
function dayEveningGreet() {
  let hour = new Date().getHours(),
      greeting;
  if (hour < 10) {
    greeting = "Good morning!";
  } else if (hour < 20) {
    greeting = "Good day!";
  } else {
    greeting = "Good evening!";
  }
  document.getElementById("demo").
  innerHTML = greeting;
}
</script>
```



Click the button to display
a time-based greeting:



Click the button to display
a time-based greeting:

Good day!

JS Conditions – switch

Use the switch statement to select one of many blocks of code to be executed

Notice that all the
cases – except the
last one – have
break statements.
Why?

```
switch (expression) {
  case n:
    //code to execute if n;
    break; // exit the statement
  case n:
    //code to execute if n;
    break; // exit the statement
  case n:
    //code to execute if n;
    break; // exit the statement
  case n:
    //code to execute if n;
    break; // exit the statement
  case n:
    //code to execute if n;
}
```


JS Conditions – the break Keyword

When JavaScript reaches a break keyword, it breaks out of the switch block

This will stop the execution of more code and case testing inside the block

When a match is found, and the job is done, it's time for a break. There is no need for more testing

A break can save a lot of execution time because it "ignores" the execution of all the rest of the code in the switch block

It is not necessary to break the last case in a switch block. The block breaks (ends) there anyway

JS Conditions – switch

```
<script>
let day;
switch (new Date().getDay()) {
  case 1:
    day = "Monday";
    break;
  case 2:
    day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case 6:
  case 0:
    day = "Weekend";
}
document.getElementById("demo").
  innerHTML = "Today is " + day;
</script>
```



Today is Monday

What's going on here?

JS Conditions – switch default Case

```
<script>
  let text;
  switch (new Date().getDay()) {
    case 5:
      text = "It's Friday!";
      break;
    case 6:
    case 0:
      text = "Weekend";
    default:
      text = "Ugh!";
  }
  document.getElementById("demo").
    innerHTML = text;
</script>
```



Ugh!

The default case does not have to be the last case in a switch block

JavaScript Loops

JS Loops

Loops can execute a block of code a number of times

Loops are handy, if you want to run the same code over and over again, each time with a different value

JavaScript supports different kinds of loops:

for - loops through a block of code a number of times

for/in - loops through the properties of an object

while - loops through a block of code while a specified condition is true

do/while - also loops through a block of code while a specified condition is true

JS Loops – for

The for loop is often the tool you will use when you want to create a loop

The for loop has the following syntax:

```
for (statement1; statement2; statement3) {
  // code to be executed;
}
```

Statement 1 is executed before the loop (the code block) starts

Statement 2 defines the condition for running the loop (the code block)

Statement 3 is executed each time after the loop (the code block) has been executed

JS Loops – for

```
for (i = 0; i < 5; i++) {
  console.log(i);
}
```



| | |
|---|------------------------------------|
| 0 | lecture.js.html:32 |
| 1 | lecture.js.html:32 |
| 2 | lecture.js.html:32 |
| 3 | lecture.js.html:32 |
| 4 | lecture.js.html:32 |

JS Loops – for/in

The JavaScript for/in statement loops through the properties of an object

```
<p>
  The for/in statement loops through<br>
  the properties of an object.
</p>
<p id="demo"></p>
<script>
  let txt = "",
      person = {fName:"Joe", lName: "Dokes", age: 25},
      p;
  for (p in person) {
    txt += person[p] + " ";
  }
  document.getElementById("demo").innerHTML = txt;
</script>
```



The for/in statement loops through the properties of an object.

Joe Dokes 25

JS Loops – while

Loops can execute a block of code as long as a specified condition is true

The while loop loops through a block of code as long as a specified condition is true

```
while (condition) {
  // code to execute
  // increment condition
}
```

If you forget to increase the variable used in the condition, the loop will never end. This will crash your browser

JS Loops – while

The JavaScript for/in statement loops through the properties of an object

```
<p id="demo"></p>
<script>
  let txt = "",
      i = 0;
  while (i < 10) {
    txt += "<br>The number is " + i;
    i++;
  }
  document.getElementById("demo").innerHTML = txt;
</script>
```



The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9

JS Loops – do/while

The do/while loop is a variant of the while loop

This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true

```
do {
  // code to execute;
  // don't forget to increment
  // the condition
} while (condition);
```

If you forget to increase the variable used in the condition, the loop will never end. This will crash your browser

JS Loops – while

The JavaScript for/in statement loops through the properties of an object

```
<p id="demo"></p>
<script>
  let txt = "",
      i = 0;
  do {
    txt += "<br> The number is " + i++;
  } while (i < 10);
  document.getElementById("demo").innerHTML = txt;
</script>
```



The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9

JavaScript Events

JS Events

HTML events are **things** that happen to HTML elements

When JavaScript is used in HTML pages, JavaScript can **react** on these events

An HTML event can be something the browser does, or something a user does

Here are some examples of HTML events:

- An HTML web page has finished loading

- An HTML input field was changed

- An HTML button was clicked

JS Events

Often, when events happen, you want to do something

JavaScript lets you execute code when events are detected

HTML allows event handler attributes, with JavaScript code, to be added to HTML elements

With single quotes:

```
<element event='some JavaScript'>
```

With double quotes:

```
<element event="some JavaScript">
```

JS Events

```
<button onclick="document.getElementById('demo').  
  innerHTML=Date()">The time is?</button>  
<p id="demo"></p>
```



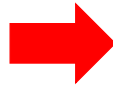
The time is?

Mon Aug 07 2017 10:21:38 GMT-0400 (Eastern Daylight Time)

JS Events

Code changes the content of its own element (using **this.innerHTML**)

```
<button onclick="this.innerHTML=Date()">
  The time is?
</button>
```



The time is?



Mon Aug 07 2017 10:26:14 GMT-0400 (Eastern Daylight Time)

JS Events

JavaScript code is often several lines long

It is more common to see event attributes calling functions

```
<p>
  Click the button to display the date.
</p>
```

```
<button onclick="displayDate()">
  The time is?
</button>
```

```
<script>
function displayDate() {
  document.getElementById("demo").
    innerHTML = Date();
}
</script>
```



Click the button to display the date.

The time is?



Click the button to display the date.

The time is?

Mon Aug 07 2017 10:40:22 GMT-0400 (Eastern Daylight Time)

JS Events

Here is a list of some common HTML events

| Event | Description |
|-------------|--|
| onchange | An HTML element has been changed |
| onclick | The user clicks an HTML element |
| onmouseover | The user moves the mouse over an HTML element |
| onmouseout | The user moves the mouse away from an HTML element |
| onkeydown | The user pushes a keyboard key |
| onload | The browser has finished loading the page |

JS – What Can JavaScript Do?

Event handlers can be used to handle, and verify, user input, user actions, and browser actions:

- Things that should be done every time a page loads

- Things that should be done when the page is closed

- Action that should be performed when a user clicks a button

- Content that should be verified when a user inputs data

JS – What Can JavaScript Do?

Many different methods can be used to let JavaScript work with events:

- HTML event attributes can execute JavaScript code directly

- HTML event attributes can call JavaScript functions

- You can assign your own event handler functions to HTML elements

- You can prevent events from being sent or being handled

JavaScript Event Listeners

JS Event Listeners

A more elegant way to react to events on a web site by creating event listeners

Removes the 'trigger' from the HTML and puts it in the JS

Similar to the ones we've seen, with some differences

JS Event Listeners

```
document.getElementById('btn1').addEventListener('click', function() {  
    //code goes here  
});
```

This appears in your JavaScript code, not the HTML

JS Event Listeners

```
<body>
  <div class='container'>
    <p id='demo'>&lt;&lt; Click here &gt;&gt;</p>
    <p id='demo2'></p>
    <span id='btn1' class='btn'>Click</span>
  </div>

  <script src='handlers.js'></script>
</body>
```

JS Event Listeners

```
document.getElementById('demo').style.cursor = 'pointer';

document.getElementById('demo').addEventListener('click', function() {
  let p = document.getElementById('demo');
  p.innerHTML = "It is so shocking to find out how many<br>" +
    "people do not believe that they can learn,<br>" +
    "and how many more believe learning to be<br>" +
    "difficult.<br>" +
    "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;- Frank Herbert, Dune";
});
```

JS Event Listeners

<< Click here >>



It is so shocking to find out how many people do not believe that they can learn, and how many more believe learning to be difficult.

- Frank Herbert, Dune

JS Event Listeners

```
<body>
  <div class='container'>
    <p id='demo'>&lt;&lt; Click here &gt;&gt;</p>
    <p id='demo2'></p>
    <span id='btn1' class='btn'>Click</span>
  </div>

  <script src='handlers.js'></script>
</body>
```

JS Event Listeners

```
document.getElementById('btn1').addEventListener('click', function() {  
  let p = document.getElementById('demo2');  
  p.innerHTML = "I must not fear. Fear is the mind-killer.<br>" +  
                "Fear is the little-death that brings total<br>" +  
                "obliteration. I will face my fear. I will<br>" +  
                "permit it to pass over me and through me. And<br>" +  
                "when it has gone past I will turn the inner<br>" +  
                "eye to see its path. Where the fear has gone<br>" +  
                "there will be nothing. Only I will remain.<br>" +  
                "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;- Frank Herbert, Dune<br>";  
});
```

JS Event Listeners

Click



I must not fear. Fear is the mind-killer.
Fear is the little-death that brings total
obliteration. I will face my fear. I will
permit it to pass over me and through me. And
when it has gone past I will turn the inner
eye to see its path. Where the fear has gone
there will be nothing. Only I will remain.
- Frank Herbert, Dune

Click

JS Event Listeners - Document

There are a variety of event listeners built in to JavaScript (in addition to 'click')

| | | | |
|--------------------------|--------------------------|-------------------------|------------------------|
| animationcancel event | drop event | pointerlockchange event | touchstart event |
| animationend event | fullscreenchange event | pointerlockerror event | transitioncancel event |
| animationiteration event | fullscreenerror event | pointermove event | transitionend event |
| animationstart event | gotpointercapture event | pointerout event | transitionrun event |
| copy event | keydown event | pointerover event | transitionstart event |
| cut event | keypress event | pointerup event | visibilitychange event |
| DOMContentLoaded event | keyup event | readystatechange event | wheel event |
| dragend event | lostpointercapture event | scroll event | |
| dragenter event | paste event | selectionchange event | |
| dragleave event | pointercancel event | selectstart event | |
| dragover event | pointerdown event | touchcancel event | |
| dragstart event | pointerenter event | touchend event | |
| drag event | pointerleave event | touchmove event | |

JS Event Listeners - Element

There are a variety of event listeners built in to JavaScript (in addition to 'click')

| | | | |
|---------------------------|------------------------|-----------------------|----------------------|
| afterscriptexecute event | DOMMouseScroll event | mousedown event | MSGestureTap event |
| auxclick event | error event | mouseenter event | MSInertiaStart event |
| beforescriptexecute event | focusin event | mouseleave event | overflow event |
| blur event | focusout event | mousemove event | paste event |
| click event | focus event | mouseout event | scroll event |
| compositionend event | fullscreenchange event | mouseover event | select event |
| compositionstart event | fullscreenerror event | mouseup event | show event |
| compositionupdate event | gesturechange event | mousewheel event | touchcancel event |
| contextmenu event | gestureend event | msContentZoom event | touchend event |
| copy event | gesturestart event | MSGestureChange event | touchmove event |
| cut event | keydown event | MSGestureEnd event | touchstart event |
| dblclick event | keypress event | MSGestureHold event | underflow event |
| DOMActivate event | keyup event | MSGestureStart event | wheel event |

JS Event Listeners - Window

There are a variety of event listeners built in to JavaScript (in addition to 'click')

| | | | |
|--------------------------|---------------------------|--------------------------|----------------------------------|
| afterprint event | DOMContentLoaded event | pagehide event | vrdisplayactivate event |
| animationcancel event | error event | pageshow event | vrdisplayblur event |
| animationend event | focus event | paste event | vrdisplayconnect event |
| animationiteration event | gamepadconnected event | popstate event | vrdisplaydeactivate event |
| animationstart event | gamepaddisconnected event | rejectionhandled event | vrdisplaydisconnect event |
| appinstalled event | hashchange event | resize event | vrdisplayfocus event |
| beforeprint event | languagechange event | storage event | vrdisplaypointerrestricted event |
| beforeunload event | load event | transitioncancel event | vrdisplaypresentchange event |
| blur event | messageerror event | transitionend event | |
| copy event | message event | transitionrun event | |
| cut event | offline event | transitionstart event | |
| devicemotion event | online event | unhandledrejection event | |
| deviceorientation event | orientationchange event | unload event | |

JS Event Listeners

This list doesn't include all of the built in events, but should illustrate the variety of things you can do with JavaScript

Questions?



Sources

https://www.w3schools.com/js/js_htmldom.asp
<https://www.w3schools.com/js/default.asp>
<http://lotrproject.com/quotes/>
<https://www.brainyquote.com/quotes/quotes//isaacasimo100104.html>
https://www.goodreads.com/author/quotes/205.Robert_A_Heinlein
<https://en.wikiquote.org/wiki/Dune>
<http://www.imdb.com/title/tt0062622/quotes>
<https://developer.mozilla.org/en-US/docs/Web/Events>

Copyrights



Presentation prepared by and copyright of John Ramsey,
East Tennessee State University, Department of
Computing . (ramseyjw@etsu.edu)



- Microsoft, Windows, Excel, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.
- IBM, DB2, DB2 Universal Database, System i, System is, System p, System ps, System x, System z, System z10, System z9, iSeries, pSeries, xSeries, zSeries, eServer, z/VM, z/OS, i3/OS, S/390, OS/390, OS/400, AS/400, S/390 Parallel Enterprise Server, PowerVM, Power Architecture, POWER6+, POWER6, POWER5+, POWER5, POWER, OpenPower, PowerPC, BatchPipes, BladeCenter, System Storage, GPFS, HACMP, RETAIN, DB2 Connect, RACF, Redbooks, OS/2, Parallel Sysplex, MVS/ESA, AIX, Intelligent Miner, WebSphere, Netfinity, Tivoli and Informix are trademarks or registered trademarks of IBM Corporation.
- Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.
- Oracle is a registered trademark of Oracle Corporation.
- HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.
- Java is a registered trademark of Sun Microsystems, Inc.
- JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.
- SAP, R/3, SAP NetWeaver, Duet, PartnerEdge, ByDesign, SAP Business ByDesign, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries.
- Business Objects and the Business Objects logo, BusinessObjects, Crystal Reports, Crystal Decisions, Web Intelligence, Xcelsius, and other Business Objects products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of Business Objects S.A. in the United States and in other countries. Business Objects is an SAP company.
- ERPsim is a registered copyright of ERPsim Labs, HEC Montreal.
- Other products mentioned in this presentation are trademarks of their respective owners.