

# JavaScript

## What is Java Script

- ❑ JavaScript is an interpreted programming language.
  - ❑ JavaScript is *the* scripting language of the Web!
  - ❑ JavaScript is the most popular scripting language on the internet
  - ❑ Runs on client side/Browsers and Server (Node.js)
  - ❑ JavaScript was designed to add interactivity to HTML pages
-

## What can a JavaScript Do?

- ❑ JavaScript gives HTML designers a programming tool
  - ❑ JavaScript can put dynamic text into an HTML page
  - ❑ JavaScript can react to events
  - ❑ JavaScript can be used to validate data
  - ❑ JavaScript can be used to detect the visitor's browser
  - ❑ JavaScript can be used to create cookies
- 

## How to Put a JavaScript Into an HTML Page

- ❑ The HTML `<script>` tag is used to insert a JavaScript into an HTML page

```
<html>
  <body>
    <script type="text/javascript">
      document.write("Hello World!")
    </script>
  </body>
</html>
```

---

## JavaScript Where To ...

- JavaScripts in the body section will be executed WHILE the page loads.

```
<html>
  <head></head>
  <body>
    <script type="text/javascript">
    </script>
  </body>
</html>
```

---

## JavaScript Where To ...

- JavaScripts in the head section will be executed when CALLED.

```
<html>
  <head>
    <script type="text/javascript">
    </script>
  </head>
</html>
```

---

## Using an External JavaScript

- The external script cannot contain the `<script>` tag!

```
<html>
  <head>
    <script src="xxx.js"></script>
  </head>
  <body>
  </body>
</html>
```

---

## JavaScript Variables

- A variable is a "container" for information you want to store
  - Rules for variable names:
    - Variable names are case sensitive
    - They must begin with a letter or the underscore character
  - **IMPORTANT!** JavaScript is case-sensitive! A variable named `strname` is not the same as a variable named `STRNAME`!
-

## Declare a Variable

- You can create a variable with the var statement:

`var strname = some value`

- You can also create a variable without the var statement:

`strname = some value`

---

## Variables in Java Script

- **Introduce with “var”**

- For global variables (!) and local variables.
- No “var” for function arguments

- **You do not declare types**

- Some people say JavaScript is “untyped” language, but technically it is “dynamically typed” language
- JavaScript is *very liberal about converting types*

- **There are only two scopes**

- Global scope Be very careful with this when using Ajax.
    - Can cause race conditions.
  - Function (lexical) scope
  - There is *not block scope as in Java*
-

## Assign a Value to a Variable

- You can assign a value to a variable like this:

```
var strname = "Hello"
```

Or like this:

```
strname = "Hello"
```

---

## JavaScript Types

- JavaScript is a *loosely typed* and *dynamic* language.
- Variables in JavaScript are not directly associated with any particular value type, and any variable can be assigned (and re-assigned) values of all types:

```
let foo = 42;    // foo is now a number  
foo      = 'bar'; // foo is now a string  
foo      = true; // foo is now a boolean
```

---

## JavaScript Types

- The set of types in the JavaScript language consists of
    - *Primitives*
    - *Objects.*
- 

## Primitives

- All types except objects define immutable values (that is, values which can't be changed).
  - For example (and unlike in C), Strings are immutable.
  - We refer to values of these types as "primitive values".
-

## Primitives

- Boolean type
  - Null type
  - Undefined type
  - Number type
  - BigInt type
  - String type
  - Symbol type
- 

## Objects

- ❑ An object is a value in memory which is possibly referenced by an identifier.
  - ❑ In JavaScript, objects can be seen as a collection of properties.
  - ❑ With the object literal syntax, a limited set of properties are initialized; then properties can be added and removed.
  - ❑ Property values can be values of any type, including other objects, which enables building complex data structures.
  - ❑ Properties are identified using key values. A key value is either a String value or a Symbol value.
-



## Objects

```
const employee = {  
  firstName: 'John',  
  lastName: 'Smith',  
  location: 'Bangalore'  
};
```

---

## Objects

### □ Some of the built-in Objects

- Date
  - Array
  - Function
  - Map
  - Set
-

## Operators and Statements

- **Almost same set of operators as Java**
    - + (addition and String concatenation), -, \*, /
    - &&, ||, ++, --, etc
    - The == comparison is more akin to Java's "equals"
    - The === operator is like Java's ==
  - **Statements**
  - Semicolons are technically optional
    - But highly recommended
    - – Consider
      - return x
      - Return  
x
  - They are not identical! The second one returns, then evaluates x. You should act as though semicolons are required as in Java.
  - **Comments**
    - Same as in Java (/\* ... \*/ and // ...)
- 

## JavaScript If...Else Statements

## Example1

```
var d = new Date()
var time = d.getHours()

if (time < 10) {
    document.write("<b>Good morning</b>")
}
```

---

## JavaScript Popup Boxes

- In JavaScript we can create three kind of popup boxes:
    - Alert box
    - Confirm box
    - Prompt box.
-

## Alert box

- ❑ An alert box is often used if you want to make sure information comes through to the user.
- ❑ When an alert box pops up, the user will have to click "OK" to proceed.

- ❑ **Syntax:**

```
alert("sometext")
```

---

## Confirm Box

- ❑ A confirm box is often used if you want the user to verify or accept something.
- ❑ When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.
- ❑ If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

- ❑ **Syntax:**

```
confirm("sometext")
```

---

## Prompt Box

- ❑ A prompt box is often used if you want the user to input a value before entering a page.
- ❑ When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.
- ❑ If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

- ❑ **Syntax:**

```
prompt("sometext", "defaultvalue")
```

---

## JavaScript Functions

- ❑ A function is a reusable code-block that will be executed by an event, or when the function is called
  - ❑ To keep the browser from executing a script as soon as the page is loaded, you can write your script as a function.
  - ❑ A function contains some code that will be executed only by an event or by a call to that function.
  - ❑ You may call a function from anywhere within the page (or even from other pages if the function is embedded in an external `.js` file).
  - ❑ Functions are defined at the beginning of a page, in the `<head>` section
-

## An Example of a function

```
function add(a, b) {  
    return a + b;  
}
```

---

## How to Define a Function

```
function functionname(var1,var2,...,varX)  
{  
    //some code ;  
    return something; // if required  
}
```

---

## Named function

- You define a function with a suitable **name**

```
function add(a, b) {  
    return a + b;  
}
```

---

## Anonymous function

- You define a function without any **name**

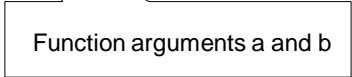
```
let adder=function (a, b) {  
    return a + b;  
}  
  
adder(2,4);
```

- The above definition of function is also known as function expression.
  - 'adder' can be passed as an argument to another function
-

## Immediately Invoked Function Expression

- JavaScript engine will invoke this function as soon as it reaches the line of definition

```
(function (a, b) {  
    return a + b;  
})(4, 6)
```



Function arguments a and b

---

## Inner Function

- Inner functions are those which are defined within the scope of the parent function.
- You can use such functions within the scope of the parent function

```
function addSquares(a, b) {  
    function square(x) {  
        return x * x;  
    }  
    return square(a) + square(b);  
}
```

---



## Important definitions in JavaScript

- ❑ Function Declaration
  - ❑ Function Statement
  - ❑ Function expression
  - ❑ First Class Function
  - ❑ Higher Order Functions
- 

## First Class Function

- ❑ A programming language is said to have First-class functions when functions in that language are treated like any other variable.
  - ❑ In such a language, a function can be **passed** as an argument to other functions, can be **returned** by another function and can be assigned as a value to a variable.
-

## First Class Functions: Examples

```
const foo = function() {  
  console.log("foobar");  
}  
foo(); // Invoke it using  
the variable  
// foobar
```

---

## First Class Functions: Examples

```
function sayHello() {  
  return "Hello, ";  
}  
  
function greeting(helloMessage, name) {  
  console.log(helloMessage() + name);  
}  
  
// Pass `sayHello` as an argument to `greeting`  
function  
greeting(sayHello, "JavaScript!");  
// Hello, JavaScript!
```

---

## First Class Functions: Examples

```
function sayHello() {  
    return function() {  
        console.log("Hello!");  
    }  
}
```

---

## Higher Order Function

- ❑ A **higher-order function** is a **function** that does at least one of the following:
    - takes one or more functions as arguments
    - returns a function as its result.
  
  - ❑ All other functions are *first-class functions*.
-

## Higher Order Function: Example

```
function sayHello() {
    return function() {
        console.log("Hello!");
    }
}

function greet() {
    return "Hello, ";
}

function greeting(helloMessage, name) {
    console.log(helloMessage() + name);
}

greeting(greet, "JavaScript!");
```

## Loops in JavaScript

- ❑ Two Types of Loops
  - for loop
  - while loop
- ❑ JavaScript Break and Continue
  - Break statement
    - ❑ Use the break statement to break the loop.
  - Continue statement
    - ❑ Use the continue statement to break the current loop and continue with the next value.

---

## JavaScript Events

- Events are actions that can be detected by JavaScript
  - We define the events in the HTML tags.
  - Examples of events
    - A mouse click
    - A web page or an image loading
    - Mousing over a hot spot on the web page
    - Selecting an input box in an HTML form
    - Submitting an HTML form
    - A keystroke
-

## Java Script Events

### □ onload and onUnload

- The onload and onUnload events are triggered when the user enters or leaves the page

### □ The onFocus, onBlur and onChange

- often used in combination with validation of form fields

```
<input type="text" size="30" id="email"
  onchange="checkEmail()">;
```

### □ **onSubmit**

- The onSubmit event is used to validate ALL form fields before submitting it

```
<form method="post" action="xxx.htm"
  onsubmit="return checkForm()">
```

---

## Java Script Events

### □ onMouseOver and onMouseOut

- onMouseOver and onMouseOut are often used to create "animated" buttons

```
<a href="http://www.jp.com" onmouseover="alert('An
  onMouseOver event');return false">
   </a>
```

---

## JavaScript Try...Catch Statement

- The try...catch statement allows you to test a block of code for errors

- **Syntax**

```
try {  
    //Run some code here  
} catch(identifier){  
    //Handle exceptions here  
}
```

---

### Example 1

```
<html>  
  <head>  
    <script type="text/javascript">  
      function message() {  
        adddler("Welcome guest!")  
      }  
    </script>  
  </head>  
  <body>  
    <input type="button" value="View message"  
      onclick="message()" />  
  </body>  
</html>
```

---

## Example 1

```
<html>
  <head>
    <script type="text/javascript">
      var txt="" function message() {
        try {
          adddler("Welcome guest!")
        } catch(err) {
          txt="There was an error on this page.\n\n"
          txt+="Error description: " + err.description + "\n\n"
          txt+="Click OK to continue.\n\n"
          alert(txt)
        }
      }
    </script>
  </head>
  <body>
    <input type="button" value="View message" onclick="message()" />
  </body>
</html>
```

---

## The Throw Statement

- ❑ The throw statement allows you to create an exception.
  - ❑ If you use this statement together with the try...catch statement, you can control program flow and generate accurate error messages.
  - ❑ **Syntax**  
 throw(exception)
  - ❑ The exception can be a string, integer, Boolean or an object.
  - ❑ Note that throw is written in lowercase letters. Using uppercase letters will generate a JavaScript error!
-



## The Throw Statement

```
<html>
<body>
<script type="text/javascript">
var x=prompt("Enter a number between 0 and 10:","")
try
{
if(x>10)
throw "Err1"
else if(x<0)
throw "Err2"
}
catch(er)
{
if(er=="Err1")
alert("Error! The value is to high")
if(er == "Err2")
alert("Error! The value is to low")
}
</script>
</body>
</html>
```

---

## JavaScript The onerror Event

- ❑ Using the onerror event is the old standard solution to catch errors in a web page
- ❑ The onerror event is fired whenever there is a script error in the page

### ❑ Syntax

```
onerror=handleErr
function handleErr(msg,url,l)
{
//Handle the error here
return true or false
}
```

---

```
<html>
<head>
<script type="text/javascript">
onerror=handleErr
var txt=""function handleErr(msg,url,l)
{
txt="There was an error on this page.\n\n"
txt+="Error: " + msg + "\n"
txt+="URL: " + url + "\n"
txt+="Line: " + l + "\n\n"
txt+="Click OK to continue.\n\n"
alert(txt)
return true
}function message()
{
addldert("Welcome guest!")
}
</script>
</head><body>
<input type="button" value="View message" onclick="message()" />
</body></html>
```

---

## JavaScript Objects Introduction

- ❑ **JavaScript is an Object Oriented Programming (OOP) language.**
  - ❑ **An OOP language allows you to define your own objects and make your own variable types**
-

## Properties

- Properties are the values associated with an object.
- the length property of the String object is used to return the number of characters in a string

```
<script type="text/javascript">  
var txt="Hello World!"  
document.write(txt.length)  
</script>
```

---

## Methods

- Methods are the actions that can be performed on objects.
- The toUpperCase() method of the String object displays a text in uppercase letters

```
<script type="text/javascript">  
var str="Hello world!"  
document.write(str.toUpperCase())  
</script>
```

---

## JavaScript String Objects

### □ The String Object

- The String object is used to manipulate a stored piece of text

---

## The Math Class

### □ Almost identical to Java

- Like Java, static methods (Math.cos, Math.random, etc.)
  - As we will see in next lecture, these are not *really static* methods, but syntax is similar to static methods in Java.
- Like Java, logs are base e, trig functions are in radians

### □ Functions

- Math.abs, Math.acos, Math.asin, Math.atan, Math.atan2, Math.ceil, Math.cos, Math.exp, Math.floor, Math.log, Math.max, Math.min, Math.pow, Math.random,

### □ Math.round, Math.sin, Math.sqrt, Math.tan

### □ Constants

- Math.E, Math.LN10, Math.LN2, Math.LOG10E, Math.PI, Math.SQRT1\_2, Math.SQRT2
-

## JavaScript Date Object

### ❑ The Date object is used to work with dates and times

### ❑ Defining Dates

- `var myDate=new Date()`

### ❑ Manipulate Dates

- set a Date object to a specific date (14th January 2010):

```
var myDate=new Date()
    myDate.setFullYear(2010,0,14)
```

- set a Date object to be 5 days into the future:

```
var myDate=new Date()
    myDate.setDate(myDate.getDate()+5)
```

---

## JavaScript Date Object

### ❑ Comparing Dates

- The following example compares today's date with the 14th January 2010

```
var myDate=new Date()
myDate.setFullYear(2010,0,14)
var today = new Date()
if (myDate>today)
    alert("Today is before 14th January 2010")
else
    alert("Today is after 14th January 2010")
```

---

## JavaScript Array Object

- The Array object is used to store a set of values in a single variable name
- Defining Arrays

- `var myArray=new Array()`

- `var mycars=new Array()`

- `mycars[0]="Saab"`  
`mycars[1]="Volvo"`  
`mycars[2]="BMW"`

- `var mycars=new Array("Saab","Volvo","BMW")`

```
var mycars=new Array(3)
mycars[0]="Saab"
mycars[1]="Volvo"
mycars[2]="BMW"
```

---

## XML Parser

**To read and update - create and manipulate - an XML document, you will need an XML parser**

## Microsoft's XML Parser

- ❑ Microsoft's XML parser is a COM component that comes with Internet Explorer 5 and higher
  - ❑ Once Internet Explorer is installed, the parser is available to scripts
  - ❑ Microsoft's XML parser supports all the necessary functions to traverse the node tree, access the nodes and their attribute values, insert and delete nodes, and convert the node tree back to XML
- 

## Microsoft's XML Parser

- ❑ To create an instance of Microsoft's XML parser with JavaScript, use the following code:

```
var xmlDoc=new ActiveXObject("Microsoft.XMLDOM")
```

- ❑ The following code loads an existing XML document ("note.xml") into Microsoft's XML parser:

```
<script type="text/javascript">  
var xmlDoc=new ActiveXObject("Microsoft.XMLDOM")  
xmlDoc.async="false"  
xmlDoc.load("note.xml")  
...  
</script>
```

---

## XML Parser in Mozilla Browsers

- Plain XML documents are displayed in a tree-like structure in Mozilla (just like IE)
- Mozilla also supports parsing of XML data using JavaScript. The parsed data can be displayed as HTML.
- To create an instance of the XML parser with JavaScript in Mozilla browsers, use the following code:

```
var xmlDoc=document.implementation.createDocument("ns","root",null)
```

---

## XML Parser in Mozilla Browsers

```
var mlDoc=document.implementation
                        .createDocument("ns","root",null)
```

- The first parameter, *ns*, defines the namespace used for the XML document
  - The second parameter, *root*, is the XML root element in the XML file
  - The third parameter, *null*, is always null because it is not implemented yet
-



## Example

- The following code loads an existing XML document ("note.xml") into Mozillas' XML parser:

```
<script type="text/javascript">  
var xmlDoc=document.implementation.  
                createDocument("", "", null);  
xmlDoc.load("note.xml");  
...  
</script>
```

---

## Loading an XML File - A Cross browser Example



**crossbrowser.txt**

---

## Loading XML Text Into the Parser

- ❑ Internet Explorer supports two ways of loading XML into a document object
  - ❑ the load() method and the loadXML() method
  - ❑ The load() method loads an XML file
  - ❑ The loadXML() method loads a text string that contains XML code
- 

### Example

```
<script type="text/javascript">
var txt="<note>"
txt=txt+"<to>Tove</to><from>Jani</from>"
txt=txt+"<heading>Reminder</heading>"
txt=txt+"<body>Don't forget me this weekend!</body>"
txt=txt+"</note>"
var xmlDoc=new ActiveXObject("Microsoft.XMLDOM")
xmlDoc.async="false"
xmlDoc.loadXML(txt)

...
</script>
```

---

## JavaScript Functions

- **Functions**
    - Basics
    - As first-class data types
    - Anonymous functions (closures)
  - **Objects**
    - Object basics
    - Namespaces (static methods)
    - JSON
    - eval
  - **Functions with variable numbers of arguments**
- 

## Getting good at Java Script

- **JavaScript is not Java**
    - If you try to program JavaScript like Java, you will *never* be good at JavaScript.
  - **Functional programming is key approach**
    - Functional programming is much more central to JavaScript programming than OOP is.
    - Java programmers find functional programming to be the single-hardest part of JavaScript to learn.
      - Because Java does not support functional programming
      - But programmers who use Ruby, Lisp, Scheme, Python, ML, Haskell, Clojure, Scala, etc. are accustomed to it
  - **OOP is radically different than in Java**
    - So different in fact, that some argue that by Java's definition of OOP, JavaScript does not have "real" OOP.
-

## Functions

### Not similar to Java

- JavaScript functions *very* different from Java methods

### Main differences from Java

- You can have global functions
    - Not just methods (functions as part of objects)
  - You don't declare return types or argument types
  - Caller can supply any number of arguments
    - Regardless of how many arguments you defined
  - Functions are first-class datatypes
    - You can pass functions around, store them in arrays, etc.
  - You can create anonymous functions (closures)
    - Critical for Ajax
    - These are equivalent
      - `function foo(...) {...}`
      - `var foo = function(...) {...}`
- 

## Functions Are First Class Data Types

- Can assign functions to variables
    - `function square(x) { return(x*x); }`
    - `var f = square;`
    - `f(5);` → 25
  - Can put functions in arrays
    - `function double(x) { return(x*2); }`
    - `var functs = [square, f, double];`
    - `functs[0](10);` → 100
  - Can pass functions into other functions
    - `someFunction(square);`
  - Can return functions from functions
    - `function blah() { ... return(square); }`
  - Can create a function without assigning it to a variable
    - `(function(x) {return(x+7);})(10);` → 17
-

## Assigning Functions to Variables

---

### Examples

```
function square(x) { return(x*x); }  
var f = square;  
square(5); → 25  
f(5); → 25
```

### Equivalent forms

```
function square(x) { return(x*x); }  
var square = function(x) { return(x*x); };
```

---

## Putting Functions in Arrays

### Examples

```
var funcs = [square, f, double];  
var f2 = funcs[0];  
f2(7); → 49  
funcs[2](7); → 14
```

### Other data structures

- Functions can also go in objects or any other category of data structure. We haven't covered objects yet, but here is a quick example:  

```
var randomObj = { a: 3, b: "Hi", c: square};  
randomObj.a; → 3  
randomObj.b; → "Hi"  
randomObj.c(6); → 36
```
-

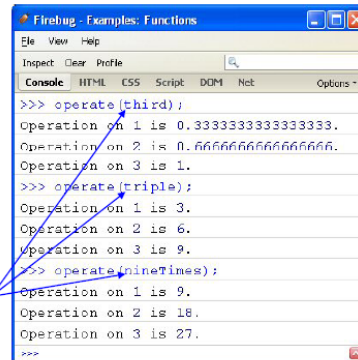
## Passing Functions to Other Functions

```
function third(x) {
    return(x / 3);
}

function triple(x) {
    return(x * 3);
}

function nineTimes(x) {
    return(x * 9);
}

function operate(f) {
    var nums = [1, 2, 3];
    for(var i=0; i<nums.length; i++) {
        var num = nums[i];
        console.log("Operation on %o is %o.",
                    num, f(num));
    }
}
```



## Objects Basics

## Basics

### Constructors

- Functions named for class names. Then use “new”.
  - No separate class definition! No “real” OOP in JavaScript!
- Can define properties with “this”
  - You must use “this” for properties used in constructors

```
function MyClass(n1) { this.foo = n1; }
var m = new MyClass(10);
```

### Properties (instance variables)

- You don’t define them separately
  - Whenever you refer to one, JavaScript just creates it

```
m.bar = 20; // Now m.foo is 10 and m.bar is 20
```

- Usually better to avoid introducing new properties in outside code and instead do entire definition in constructor

### Methods

- Properties whose values are functions
- 

## Object: An Example

```
function Circle(radius) {
    this.radius = radius;

    this.getArea =
        function() {
            return(Math.PI * this.radius * this.radius);
        };
}

var c = new Circle(10);
c.getArea(); // Returns 314.1592...
```

---

## The **prototype** Property

### In previous example

- Every new Circle got its own copy of radius
  - Fine, since radius has per-Circle data
- Every new Circle got its own copy of getArea function
  - Wasteful, since function definition never changes

### Class-level properties

- `Classname.prototype.propertyName = value;`

### Methods

- `Classname.prototype.methodName = function() {...};`
  - Just a special case of class-level properties
- This is legal anywhere, but it is best to do it in constructor

### Pseudo-Inheritance

- The prototype property can be used for inheritance
    - Complex. See later section on Prototype library
-