Overview

- Overview of JavaScript
- Object Orientation
- Syntactic Characteristics
- Primitives, Operations and Expressions
- Math, Number, String and Date objects
- Screen Output
- Control Statements, Arrays and Functions

Origins of JavaScript

- Originally developed by Netscape, as LiveScript
- Became a joint venture of Netscape and Sun in 1995, renamed JavaScript
- Now standardized by the European Computer Manufacturers Association as ECMA-262
- An HTML-embedded scripting language
- We'll call collections of JavaScript code *scripts*, not programs

What is JavaScript ?

A scripting language is a lightweight programming language. ***JvaScript is a scripting language*** designed primarily for adding interactivity to Web pages and creating Web applications.

Client-side JavaScript programs, or scripts, can be embedded directly in HTML source of Web pages. (There is also server side scripting language)

Object Orientation

- JavaScript is NOT an object-oriented programming language
  - Rather object-based or
  - Prototype based
- Does not support class-based inheritance
  - Cannot support polymorphism
- JavaScript objects are collections of properties, which are like the members of classes in Java
  - Data and method properties
- JavaScript has primitives for simple types
- The root object in JavaScript is Object – all objects are derived from Object

JavaScript is an object-oriented language with ***prototypal inheritance***.

- The language supports several built-in objects, and programmers can create or delete their own objects.
- Prototypal inheritance makes JavaScript very different from other popular programming languages such as C++, C#, or Java featuring *classes* and *classical inheritance*.
- JavaScript does not have classes in the C++ or Java sense.
- In JavaScript, objects can inherit properties directly from each other, forming the object prototype chain.

JavaScript is an interpreted language, with ***optional JIT-compilation*** support

- JavaScript was a purely *interpreted language*. This means that scripts execute without preliminary *compilation*, i.e. without conversion of the script text into system-dependent machine code.

- The user's browser *interprets* the script, that is, analyzes and immediately executes it. In modern implementations, JavaScript code may be either interpreted or compiled using a *just-in-time* (JIT) compiler.

- At run time, the browser decides whether (parts of) script code should be JIT-compiled for better performance. This makes JavaScript significantly *faster* and therefore more suitable for complex performance-demanding Web applications. Recent versions of all popular browsers have JavaScript JIT-compilers.

JavaScript and Java

- JavaScript and Java are only related through syntax
- JavaScript is dynamically typed
- Java Script’s support for objects is very different
- JavaScript is interpreted
  - Source code is embedded inside XHTML doc, there is no compilation

Uses of JavaScript

- Transfer of some load from server to client
- User interactions through forms
  - Events easily detected with JavaScript
  - E.g. validate user input
- The Document Object Model makes it possible to create dynamic HTML documents with JavaScript

Syntactic Characteristics

- Identifier form: begin with a letter or underscore, followed by any number of letters, underscores, and digits
  - Case sensitive
- 25 reserved words, plus future reserved words
- Comments: both // and /* … */
- Scripts are usually hidden from browsers that do not include JavaScript interpreters by putting them in special comments

```
<!--
-- JavaScript script –
//-->
```

- Semicolons can be a problem
  - They are “somewhat” optional
  - Problem: When the end of the line can be the end of a statement – JavaScript puts a semicolon there

JavaScript reserved words

Break,delete,function,return,typeof,case,do,if,switch,var,catch,else,in,this.void,continue,finally.instanceof,throw,while,default,for,new,try,with

Embedding in XHTML docs

- Either directly, as in

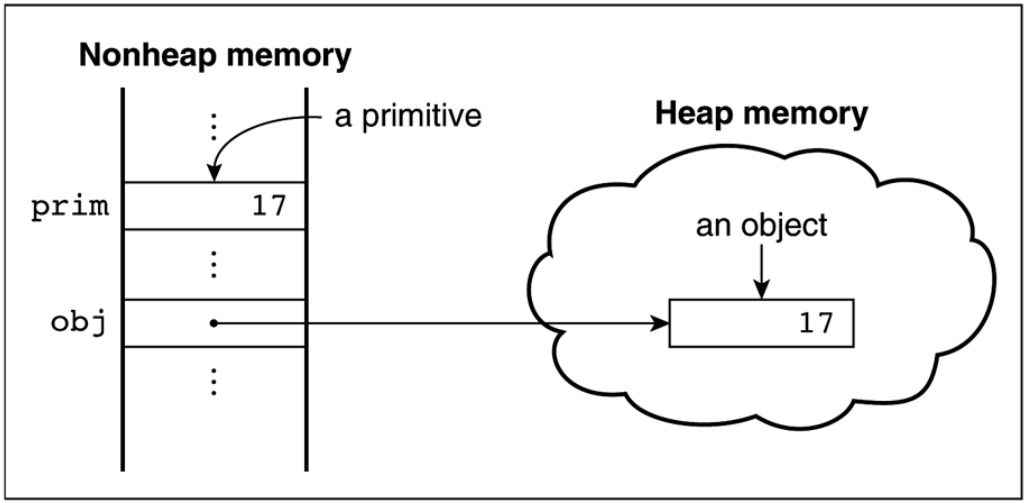
```
<script type = “text/javascript”>
    -- JavaScript script –
</script>
```

- Or indirectly, as a file specified in the `src` attribute of `<script>`, as in

```
<script type = “text/javascript”
    src = “myScript.js”>
</script>
```

Primitives

- All primitive values have one of the five types: Number, String, Boolean, Undefined, or Null
- Number, String, and Boolean (**wrapper objects)**
- The purpose of wrapper objects is to provide properties and methods that are convenient for use with values of the corresponding primitive type.



Primitives

- All numeric values are stored in double-precision floating point
- String literals are delimited by either ' or "
- Boolean values are true and false
- The only Null value is `null`
- The only Undefined value is `undefined`

Declaring Variables

- JavaScript is dynamically typed – any variable can be used for anything (primitive value or reference to any object)
- The interpreter determines the type of a particular occurrence of a variable
- Variables can be either implicitly or explicitly declared

```
var sum = 0,
    today = "Monday",
    flag = false;
```

Numeric Operators

Operator	Associativity
`++`, `--`, unary `-`	Right (though it is irrelevant)
`*`, `/`, `%`	Left
Binary `+`, binary `-`	Left

The first operators listed have the highest precedence.

`Math` and `Number` Objects

- The `Math` Object provides `floor`, `round`, `max`, `min`, trig functions, etc.
  - e.g., `Math.cos(x)`
- The `Number` Object has some useful properties

Property	Meaning
`MAX_VALUE`	Largest representable number
`MIN_VALUE`	Smallest representable number
`NaN`	Not a number
`POSITIVE_INFINITY`	Special value to represent infinity
`NEGATIVE_INFINITY`	Special value to represent negative infinity
`PI`	The value of π

`String` Object

- The number of characters in a string is stored in the `length` property

```
var str = “George”;
var len = str.length;
```

- Common methods:

Method	Parameters	Result
`charAt`	A number	The character in the `String` object that is at the specified position
`indexOf`	One-character string	The position in the `String` object of the parameter
`substring`	Two numbers	The substring of the `String` object from the first parameter position to the second
`toLowerCase`	None	Converts any uppercase letters in the string to lowercase
`toUpperCase`	None	Converts any lowercase letters in the string to uppercase

`Date` Object

- Create one with the Date constructor (no params)

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

- Local time methods of Date:
  - toLocaleString – returns a string of the date
  - getDate – returns the day of the month
  - getMonth – returns the month of the year (0 – 11)
  - getDay – returns the day of the week (0 – 6)
  - getFullYear – returns the year
  - getTime – returns the number of milliseconds since January 1, 1970
  - getHours – returns the hour (0 – 23)
  - getMinutes – returns the minutes (0 – 59)
  - getMilliseconds – returns the millisecond (0 – 999)

Screen Output

- JavaScript models the HTML document with the `Document` object
- The model for the browser display window is the `Window` object
  - The `Window` object has two properties, `document` and `window`, which refer to the `Document` and `Window` objects, respectively
- The `Document` object has a method, `write`, which dynamically creates content
  - The parameter is a string, often concatenated from parts, some of which are variables

```
document.write("Answer: “, result, "<br>");
```

  - The parameter is sent to the browser, so it can be anything that can appear in an HTML document (any HTML tags)

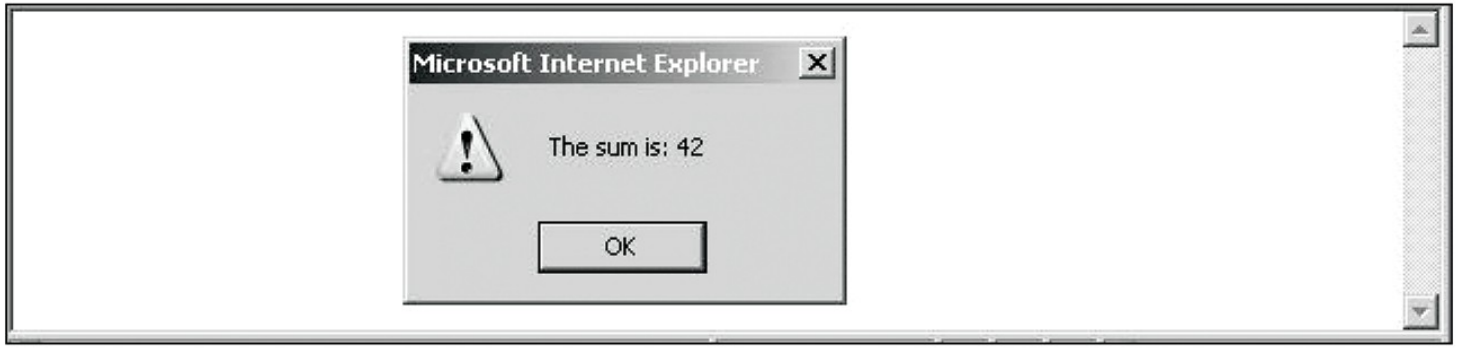
Screen Output

- The `Window` object has three methods for creating dialog boxes

1. **Alert**

```
alert(“The sum is:” + sum + ”\n");
```

   - Parameter is plain text, not HTML
   - Opens a dialog box which displays the parameter string and an OK button



Screen Output

1. **Confirm**

```
var question = confirm("Do you want
to continue this download?");
```

- Opens a dialog box and displays the parameter and two buttons, OK and Cancel
- Returns a Boolean value, depending on which button was pressed (it waits for one)

Screen Output

1. **Prompt**

```
prompt("What is your name?", “ ");
```

- Opens a dialog box and displays its string parameter, along with a text box and two buttons, OK and Cancel
- The second parameter is for a default response if the user presses OK without typing a response in the text box (waits for OK)

http://www.cs.nott.ac.uk/~bnk/WPS/roots.html

Conditionals

- Selection statements – “`if`” and “`if…else`“

```
if (a > b)
  document.getElementById(“x”).innerHtml=(“a is
greater than b <br>”);
else {
  a = b;
  document.getElementById(“x”).innerHtml(“a was
not greater than b, now       they are equal
<br>”);
}
```

- The `switch` statement

Loops

- `while` *(control_expression)*
  statement or compound stmt

- `for` *(init; control; increment)*
  statement or cmpnd stmt
  - init can have declarations, but the scope of such variables is the whole script

- `do` *statement or compound*
  `while` *(control_expression)*

Functions

A function is a piece of code that sits dormant until it is referenced or called upon to do its "function".

```
Ex:function display( ) {
    alert("Hello World")
}
<input type="button" onclick="display()"
   value="Click">
```

Functions

```
function function_name([formal_parameters]) {
  -- body –
}
```

- Return value is the parameter of `return`
  - If there is no return or if return has no parameter, undefined is returned
- We place all function definitions in the head of the HTML document
  - Calls to functions appear in the document body
- Variables explicitly declared in a function are local

Functions – parameters

- Parameters are passed by value, but when a reference variable is passed, the semantics are pass-by-reference
- There is no type checking of parameters, nor is the number of parameters checked
  - excess actual parameters are ignored, excess formal parameters are set to undefined
- All parameters are sent through a property array, `arguments`, which has the `length` property

Object Creation and Modification

Objects can be created with `new`

- The most basic object is one that uses the `Object` constructor, as in

```
var myObject = new Object();
```

- The new object has <u>no properties</u> - a blank object

- Properties can be added to an object, any time

Object Creation and Modification

```
var myAirplane = new Object();
    myAirplane.make = "Cessna";
    myAirplane.model = "Centurian";
```

- Objects can be nested, so a property could be itself another object, created with `new`

- Properties can be accessed by dot notation , as in

```
var property1 = myAirplane["model"];
```

- If you try to access a property that does not exist, you get `undefined`

- Properties can be deleted with `delete`, as in

```
delete myAirplane.model;
```

- *Another Loop Statement*

  - `for (`identifier in object`)` statement or compound

```
for (var prop in myAirplane)
  document.write(myAirplane[prop] + "<br />");
```

Arrays

- Array elements can be primitive values or references to other objects
- Array objects can be created in two ways, with `new`, or by assigning an array literal

```
var myList = new Array(24, "bread", true);
var myList2 = new Array(24);
var myList3 = [24, "bread", true];
```

- Length is **dynamic** - the `length` property stores the length
  - length property is writeable

```
myList.length = 150;
```

Methods In Array

Pop( )-The pop() method removes the last element of an array, and returns that element.

Push( )-The push() method adds new elements to the end of an array, and returns the new length.

Reverse ( )-The reverse() method reverses the order of the elements in an array (makes the last element first, and the first element last).

Splice( )- The splice() method adds and/or removes elements to/from an array, and returns the removed element(s).

Syntax:

array.splice(index,howmany,element1,.....,elementX)

Shift ( )-Removes the first element of an array, and returns that element

Unshift ( )-Adds new elements to the beginning of an array, and returns the new length

Constructors

Javascriipt constructors are special methods that create and initialize the properties.

```
Function car(new_make, new_model, new_year){
this.make=new_make;
this.model= new_model;
This.year=new_year;
```

```
my_car= new car(“Ford”,”SVT”,”2000”)
Function dispaly_car()
{
document.getElementById(“CAR”).innerHTML=(“car make:”this.make,”<br/>);
```

Output:

Car make:Ford

Pattern Matching and Regular Expressions

Two approaches

1. RegExp object
2. methods of the string object

Metacharacters:

\| ( ) [ ] { } ^ $ * +?

*n- Matches any string that contains zero or more occurrences of n

+n-Matches any string that contains at least one n

?n-Matches any string that contains zero or one occurrences of n

n$-Matches any string with n at the end of it

^n- Matches any string with n at the beginning of it

i Perform case-insensitive matching.

m Perform multiline matching

g Performs a global match that is, find all matches rather than stopping after the first match.

Examples:

“snow”=> /snow./

/5\.6/ => matches 5.6

'a','b','c'=> [abc]

'a' to 'k'=>[a-k]

Predifined character classes

\d - this escape character represents any digit and is equivalent to [0-9]

\D - this escape character represents everything except digits and corresponds to [^0-9]

\w - this escape character represents any "word" character and corresponds to [a-zA-Z0-9_]

\W - this escape character matches anything that isn't a letter, number or underscore

\s - this escape character matches any whitespace (spaces, tabs,linefeeds, carriage returns, nulls)

\S - this escape character represents anything that isn't whitespace

/\d\.\d\d/ =>Matches a digit,followed by a period,

Summary

- Overview of JavaScript
- Object Orientation
- Syntactic Characteristics
- Primitives, Operations and Expressions
- Math, Number, String and Date objects
- Screen Output
- Control Statements, Arrays and Functions