

JavaScript

Introduction

1

JavaScript

- background history and origins
 - strengths and weaknesses
 - alternatives to JavaScript
 - JavaScript in the browser
 - JavaScript platforms
- JavaScript and HTML
- syntax
 - data types
- statements
 - if/then
 - iteration
- Date objects
- Array
- functions
- regular expressions

2

JavaScript origins

- Netscape and Sun Microsystems developed a scripting language named JavaScript in 1995 to add new functionality within web pages
- originally designed for the Netscape Navigator browser
- Brendan Eich, now CTO of Mozilla, developed JavaScript
 - originally named Mocha, then LiveScript
 - Netscape changed its name to JavaScript purely for marketing due to growing Java popularity

3

JavaScript origins

- JavaScript is not Java, a complex programming language designed for diverse computing purposes
 - Java uses static binding not dynamic binding
- all major browsers support JavaScript, now one of the most popular web languages
- JavaScript was originally designed with a simpler Java-like syntax just for browsers
- many language syntax similarities with Java and C
- JavaScript is an implementation of the ECMAScript language standard (ECMA International organization – European Computer Manufacturers Association)

4

JavaScript origins

- prior to JavaScript, web pages used server-side programs (CGI¹) to handle user interaction with forms, buttons, and menus -- an internet connection to web server must be maintained
- with client-side JavaScript the actions of the user are handled by the browser not the web server – means more information in the web page to download from server but overall faster user experience

5

JavaScript origins

- browsers equipped with a JavaScript engine interpret then execute the JavaScript code as required
- early JavaScript uses include handling user events like mouse click, hover over a button, and verify data entry in a form
- JavaScript is officially managed by Mozilla Foundation
- "JavaScript" is a trademark of Oracle Corporation

6

JavaScript - strengths

- quick development
 - no special creation software required
 - fast test and modify cycle
 - many free resources and frameworks available
- easy to learn
 - doesn't share the more complex syntax of Java
 - object oriented structure
- platform independence – all operating systems support it
- interfaces well with the DOM (Document Object Model)
- the most popular web development language
 - basis for JSON, jQuery, and AJAX technologies
- small overhead for browser resources
 - fast download of HTML and JavaScript script, even faster if they are in separate files
- JavaScript code files can be easily shared

7

JavaScript - weaknesses

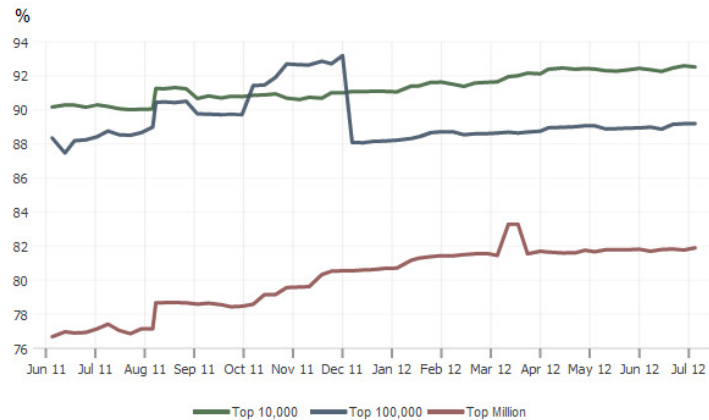
- parts of the language can be difficult to master
 - concept of closures, anonymous methods in JavaScript
 - not as well suited for team development as other languages
- rendering varies by browser
 - browsers employ different JavaScript engines resulting in inconsistent functionality and interface
- no JavaScript code hiding
 - JavaScript visible within the HTML page
 - consensus has become JavaScript scripts are essentially 'freeware'
 - JavaScript executing within browser could potentially have malicious code exploits on the user's system
- users have option to disable JavaScript in browser
 - prevent storage of cookies, pop-ups, and other functionality
- search engines may ignore HTML pages containing a lot of JavaScript code
 - Web developers can isolate all the JavaScript code into a separate .js file
- JavaScript always stops running at the first sign of an error
 - Even if you have multiple errors in the JavaScript, only the first one is flagged

8

JavaScript - trends

Javascript Usage Trends

JavaScript is a scripting language most often used for client-side web development. Its proper name is ECMAScript, though "JavaScript" is much more commonly used. The site uses JavaScript.

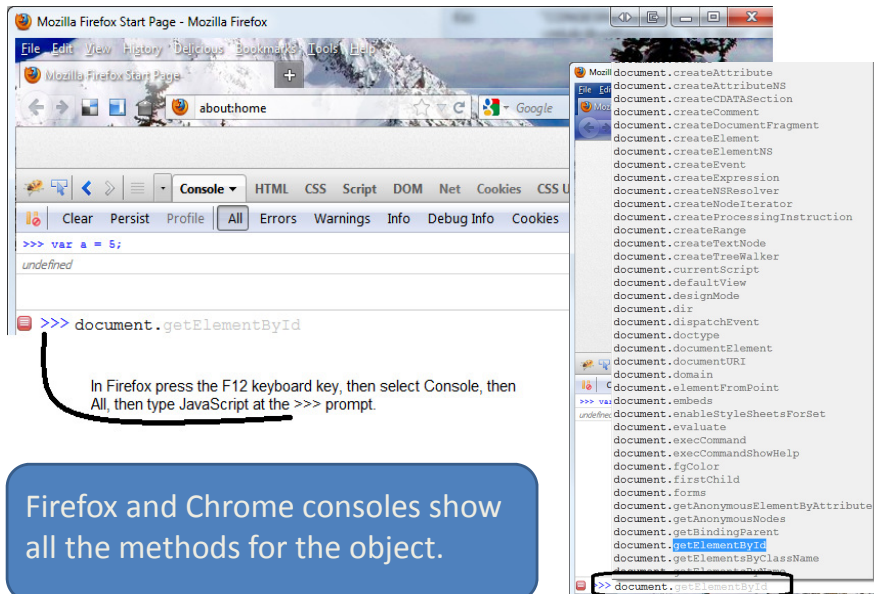


<http://trends.builtwith.com/docinfo/Javascript>

Alternatives to JavaScript

- Google Dart
 - <http://www.dartlang.org/docs/spec/latest/dart-language-specification.html>
 - supported only by Google Chrome (as of May 2012)
 - syntax similar to C
- CoffeeScript
 - <http://www.coffeescript.org>
 - transcompiles to JavaScript
 - language adds syntactic sugar from Ruby, Python, and Haskell
- Haxe
 - <http://haxe.org>
 - compiles to Adobe Flash, PHP, or JavaScript
- Opa
 - <http://opalang.org>
 - Can be used for client-side and server-side scripting
 - Influenced by Ocaml and Erlang programming languages
- Google Web Toolkit, RubyJS, Pyjamas
 - Use Java language to manage web front end applications
 - RubyJS is the Ruby language implementation, Pyjamas is the python language implementation

JavaScript – browser console



JavaScript and HTML

- HTML tags `<script>` `</script>` contain the JavaScript portion within the HTML file
- attribute "type" identifies the MIME type of the script (usually `text/javascript`)
- as of HTML 5, the "type" is optional and will default to `text/javascript` if not provided
- attribute `language="JavaScript1.8"` is deprecated

```
<script type="text/javascript">  
  
    JavaScript source script appears in here  
  
</script>
```

13

JavaScript and HTML

- hiding scripts from older browsers (pre-IE 6 vintage) which do not support JavaScript, use the HTML comment element – now obsolete `<!-- -->`

```
<script type="text/javascript">  
<!-- Hide the script from some browsers  
  
JavaScript program code ...  
  
// Stop hiding from other browsers -->  
</script>
```

single line comments in
JavaScript use `//`
notation

14

JavaScript and HTML

- where to place the JavaScript code within the HTML document?
- JavaScript programs can be included anywhere in the header or body of the HTML
- if there is JavaScript to execute prior to the HTML page rendering, then place it in the HTML header
- JavaScript can be defined anywhere within the HTML body if JavaScript functionality is appropriate for that part of the HTML
 - form validation of user entered data , mouse hover, or DHTML (dynamic HTML)

15

JavaScript and HTML

- longer or complex scripts can be placed in a separate text file which must have a `.js` file extension
- usually the scripts that affect page layout are defined within the head element and external scripts (Google analytics, e.g.) at the bottom of the body element (just before `</body>`) to improve the page rendering time
- JavaScript code in the `.js` file cannot have the `<script>` element

```
<script type="text/javascript"  
    src="http://www.you.com/myScript.js">  
</script>  
<script type="text/javascript"  
    src="js/myJSlib.js">  
</script>
```

16

JavaScript - sample 1

- following page shows HTML having an embedded JavaScript script in the body element using DHTML to write some text (Have a nice day!) to the browser window.
- JavaScript is updating the HTML page content via the DOM object document

`document.writeln`

17

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

<body>
  This is sample DHTML JavaScript:
  <script type="text/javascript">

    // Display a greeting message.
    document.writeln("Have a nice day!<br />");

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

18

JavaScript syntax

- basic unit is one-line statement or expression followed by a semicolon (not mandatory but strongly recommended)
- `document.writeln("...");`
 - JavaScript command invokes the DOM's document object method `writeln()`
- in JavaScript, as in Java, everything is *case-sensitive*
 - use *document* NOT *Document* or *DOCUMENT*
 - use *writeln* NOT *WRITELN* or *WriteLN*

19

JavaScript syntax

- Terms:
 - method
 - name of a function associated with an object
 - e.g. write and writeln are methods of object document
 - parameter
 - In the definition of the method or function, the placeholder values passed into the method or function
 - e.g. function add(n) { return n+1; } // n is parameter
 - argument
 - The actual values used in the invocation of the method or function
 - e.g. document.write("Hello "); // "Hello" is argument
 - when more than one is used, parameters and arguments are separated by commas

20

JavaScript syntax

- JavaScript layout is free-format
 - it does not matter how you format your JavaScript with white spaces (tabs, new lines)
 - multiple statements on one line separated by ;
 - readability is key if you are maintaining the JavaScript code for development
 - many third party JavaScript libraries are provided in *minimized* form to speed up download (all newlines and unnecessary spaces stripped out) and may obfuscate the JavaScript code (hinder reverse engineering)
 - there are JavaScript code formatters which make JavaScript more easily readable to humans
 - <http://javascript.crockford.com/code.html>

21

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

document.writeln ( "This " ) ;
    document.writeln(
        " is "
    );
document.writeln( " a " )
;
    document.writeln( "line \
. \
" ) ;

</script>
```

Free-format demo –
do not write
JavaScript like this!

If text spans more than
one line, use the
backslash \ to continue.

22

JavaScript tools

- JSLint is a JavaScript program that checks for problems in your JavaScript programs (improves code quality)
 - jslint.com
- YUI Compressor minimizes JavaScript and CSS
 - developer.yahoo.com/yui/compressor
- Dojo Toolkit allows you to use and build custom web page widgets for many platforms
 - dojotoolkit.org

23

JavaScript comments

- use comments in JavaScript to explain the code purpose and make it human readable.
- use // for one line comment and /* */ for multiline

```
// Use the numeric sort function.
function s(a,b) {
    return (a-b);
}
/*
    This code will write to a heading.
*/
document.getElementById("theHeading").innerHTML
= "This is the beginning.";
```

24

JavaScript methods

- `document.writeln("Hello
");`
 - outputs the text `Hello
` to the browser, which interprets it as HTML information
 - `writeln` is one of many *methods* associated with the object `document`
- all methods are functions in JavaScript
- in JavaScript, methods are called by combining the object name with the method
 - `objectname.methodname`
- if the object name is omitted, the *window* object is assumed (e.g. `alert` is `window.alert`)

25

JavaScript methods

- data that the method needs to perform is provided as an **argument** within the parenthesis:
`document.write("Welcome !");`
`document.writeln("Have a great day!");`
- script container does not affect the HTML structures where it occurs, so any format tags or elements in the HTML file will affect the text produced by `write()` and `writeln()` methods

26

```
<html>
<head>
<title>JavaScript Sample 2</title>
</head>

<body>
Here is a sample JavaScript <b>
<script type="text/javascript">

// Display a message
document.writeln("This text appears as bold. ");
document.writeln(" </b>");

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

27

JavaScript - prompt

- JavaScript can interact directly with the user
- simplest way is with the JavaScript `prompt()` method
- `prompt` displays its first argument as the prompt text
- optional second argument is displayed as the default value within the dialog box
- empty string is returned if user clicks OK without providing any text

28

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

var name = prompt("Enter your name:",
                  "visitor"));

document.write("Welcome " + name );

var sign = prompt("What is your zodiac sign?");

document.write("<br />");

document.write("Your sign is " + sign + ".");
</script>
```

29

JavaScript - prompt

- the `prompt` method doesn't need to be prefixed by `document.` because it is a method of the `window` object
- if the object name is missing, then `window` object is assumed
 - e.g. JavaScript functions `parseInt`, `parseFloat`, `isNaN` do not require to be prefixed by `window`.

30

JavaScript - alert

- `alert` dialog box
- useful to show some information in a dialog box
- `alert("Click OK to continue.");`
- useful to point out
 - incorrect information in a form
 - invalid result from a calculation
 - other immediate messages

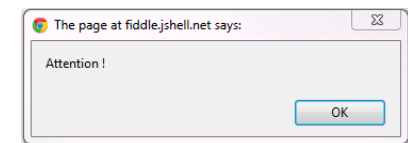
31

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

document.write('');

alert("Attention !");

</script>
```



32

JavaScript - variables

- variable names are case sensitive and must start with a letter, dollar sign, or underscore; subsequent characters can be digits 0-9; no reserved JavaScript keywords* allowed
- best practice: variable name starts with a-z
- valid JavaScript variable names:
a rangeRow x1 p_input salary2012
- invalid JavaScript variable names:
a# @tag 4H X factor true
- * https://developer.mozilla.org/en/JavaScript/Reference/Reserved_Words

33

JavaScript - variables

- keyword `var` declares variables
- subsequent use of `var` for the same variable within the same script block is unnecessary

```
var a;  
var selection;  
var b, c, d;
```

34

JavaScript - constants

- a read-only named constant is created with the `const` keyword
- same name rules as for variables
- constants cannot change value or be re-declared
- cannot use same name as an existing function or variable

```
const g = 10.5;
```

35

JavaScript - assignment

- the single equals sign `=` is the assignment operator e.g. *variable = expression*;
 - expression on the right is evaluated and the variable name on the left represents that value

```
var a = 0; // declare variable a having value 0  
a = 100+1; // variable a now has value 101  
a = "cat"; // variable a now has value "cat"  
var b = 0, c = true, d = "atom"; // 3 variables  
a = b; // variable a now has value zero
```

36

JavaScript – scope rules

- in general, always preface the declaration of new variables with the `var` keyword
- if you declare a new variable without the `var` keyword (*implicit declaration*), you may be accidentally changing the value of the same variable name found in a higher scope...but you are permitted to use `delete` statement on it ... not so if you use the `var` keyword
- more about this later in the scoping section in functions

37

JavaScript - block

- a block statement is used to group one or more statements within braces `{ }`
- commonly used with control flow as in loops

```
{  
  statement_1;  
  statement_2;  
  ...  
  statement_n;  
}
```

38

JavaScript - block

- JavaScript does **not** have block scope. Variables declared within a block are scoped to the containing function or script, and any assignment of values to them continue beyond the block itself. (v1.7 JavaScript introduces a `let` keyword which changes this – to be discussed later)

```
var a = 1;  
{  
  var a = 5;  
}  
// variable a is 5
```

39

JavaScript - variables

- multiple variables may be declared with one `var` statement – each separated by a comma
`var a = 0, b, c = 100, d = "blue sky", e = a;`
- this practice is slightly more execution efficient than declaring each variable with a separate `var` but not as maintainable
 - potentially, an error will occur if you remove a declared variable and the comma separator
 - e.g `var a = 0 , b d = "blue sky", e = a;`

40

JavaScript – data types

- JavaScript provides five primitive data types
 - numeric as in 0, -21, and 32.62
 - strings as in "Hello" and 'There'
 - Boolean (logical) either true or false
 - **null** special keyword for a nothing value; null is primitive and case-sensitive (not NULL or Null)
 - **undefined** for something not yet assigned a value or an unknown variable; also primitive

41

JavaScript - numeric

- an integer number is a sequence of digits
 - range is -2^{53} to 2^{53} (-9007199254740992 to 9007199254740992 inclusive)
 - base 10 integers (decimal) do not start with a zero
 - base 8 integers (octal) start with a zero (deprecated)
 - base 16 integers (hexadecimal) start with 0x

```
var a = 0100; // a is 64
var b = 100;  // b is 100
var c = 0x010; // c is 16

var d = 0x3a - 0200; // d is -70
var e = -073 - 0x0b; // e is -68
```

42

JavaScript - numeric

- floating point literals
 - floating-point literals must have at least one digit and either a decimal point or "e" (or "E")
 - range is 5e-324 to 1.797e308
 - JavaScript keyword: Infinity or -Infinity
 - Number.POSITIVE_INFINITY, Number.NEGATIVE_INFINITY and Number.MAX_VALUE, Number.MIN_VALUE

```
var a = 10.010101;
var b = -0.99;
var c = 1.45E10;
var d = 2e-2;
var bigNum = 2/0; // bigNum is Infinity
```

43

JavaScript - string

- strings store a piece of text
- JavaScript has two kinds of strings: **primitives** and **objects**
- **primitives**: can use JavaScript String() or assignment

```
var txt = String("Hello");
var txt = "Hello";
```
- **objects**: use **new** String()

```
var txt = new String("Hello");
```
- use **primitive** form unless object form is required.

44

JavaScript - string

- string length displayed using length method
`var txt_len = "hello".length; // txt_len is 5`
- empty string "" has a length of zero
- special characters such as `"\"` and `backspace`, `newline`, `tab`, `carriage return` can be defined within a string this way: `"\b"` `"\"`, `"\n"`, `"\t"`, `"\r"` respectively
`var t = "He said, \"Welcome\".";`

45

JavaScript - string

- concatenation operators are `+` and `+=`
`"Welcome to " + "my house"` makes the string `"Welcome to my house"`

`welcome += " Thank-you."` adds the string `"Thank-you."` to the end of the string variable named `welcome`

also, `string1.concat(string2)` method

```
var n = "abc";  
var t = n.concat("xyz");  
// t is "abcxyz"; n is "abc"
```

46

JavaScript - string

- access an individual character within a string in two ways, using the `charAt` method or as an array (first character is index zero)
 - `"mouse".charAt(1)` is `"o"`
 - `"mouse"[1]` is `"o"`

```
var pet = "mouse";  
var c = pet.charAt(1); // c is "o"  
c = pet[1];           // c is "o"
```

47

JavaScript - string

- `substr` method returns a portion of a string
 - `string.substr(start_index, length)` `length` is optional but if not provided, extract characters until end of string

```
var answer = "quick";  
  
var n1 = answer.substr(1, 2); // ui  
  
var n2 = answer.substr(2);    // ick  
  
var n3 = answer.substr(-1);   // k
```

https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/String#Properties_2

48

JavaScript - string

- `replace` method substitutes one substring with another

– `string.replace(search_string, new_string)`

```
var t = "white car with white seat";

var n = t.replace("white", "blue");
var p = t.replace(/white/g, "red");
// n is "blue car with white seat"
// p is "red car with red seat"
// t is "white car with white seat"
```

49

JavaScript - string

- `toLowerCase` and `toUpperCase` convert the string's case

– these two methods require no arguments

```
var city = "Victoria, BC";

var n1 = city.toLowerCase(); // victoria,bc

var n2 = city.toUpperCase(); // VICTORIA,BC
// city is still
// Victoria, BC
```

50

JavaScript - string

- string `"null"` is not the same as `null`
- string `"undefined"` is not the same as `undefined`
- string `""` is not the same as `null` or `undefined`

51

JavaScript - boolean

- boolean values are either `true` or `false`
- double equals operator `==` tests if two operands represent the same value (but not the same *type*)
- triple equals operator `===` tests if two operands represent the same value **and** the same type
- non-zero numeric values evaluate to `true`
- `null`, `undefined`, `NaN`, and `""` evaluate to `false`

```
var a = true;
var b = false;
var c = (1 == 1); // c is true
var d = (a = 2); // d is true, a is 2
var e = (1 == "1"); // e is true
var f = (1 === "1"); // f is false
```

52

JavaScript - typing

- JavaScript is a *dynamically typed* programming language
 - variables are not defined by data type at declaration but by their values (or 'literals')
- the type of a literal is defined based on context (run-time)
- when combining literals of different types, the first type is used
- Java and C are *statically typed* – the type of the variable is set at compile time permanently

53

JavaScript - typeof

- the typeof operator is unary – use of () optional
 - e.g. typeof("pumpkin"), typeof(563), typeof(true), typeof(null), or typeof "squash"
 - returns **type** of the operand: "number", "string", "boolean", "object", "function", undefined, "xml"

```
var a = "cherry";  
var a_type = typeof(a); // a_type is "string"  
var b = 3.14;  
var b_type = typeof(b); // b_type is "number"  
var c;  
var c_type = typeof c; // c_type is undefined  
var d = null;  
var d_type = typeof d; // d_type is "object"
```

54

JavaScript – dynamic typing

```
var a = 99;  
var b = "Ninety nine";  
var c = 100 + 100; // c is 200  
var d = ( a < 100 ); // d is true  
var e = d && ( c > 100 ); // e is true  
a = e; // a is true  
var f = "100" + 10; // f is 10010  
var g = "100" - 10; // g is 90
```

55

JavaScript – weak typing

- JavaScript is also weakly typed
 - no restrictions on use of operators (such as the plus sign) involving values of different data types
- JavaScript rule: when you use + with a number and a string in any order you get a string result

```
var a = 100;  
var b = "+100";  
var sum = a + b; // sum is "100+100" not 200  
sum = parseInt(a) + parseInt(b); // sum is 200
```

56

JavaScript - casting

- JavaScript data type examples
 - "Count to " + 10 is "Count to 10"
 - and 2.5 + "10" is "2.510"
- `parseInt()` and `parseFloat()` JavaScript functions cast values to a new type :
 - `parseInt("12")` returns the integer 12
 - `parseFloat("33.23")` returns 33.23
 - `parseInt("23.66")` returns 23
 - `parseInt(undefined)` and `parseInt(null)` returns NaN (not a number)
 - optional second argument is the radix (10 is default, 16, or 8 but that is deprecated) `parseInt("0xaa", 16)` is 170 decimal.
- see <http://jsfiddle.net/Stevelang/vpenh/>

57

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

var answer = 99;

answer = "Ninety nine ";

var question = "What is 9 times 11? " + answer;

document.write(question + "<br />");

question = answer + " is 9 times what number?";

document.write(question);

</script>
```

58

JavaScript - expressions

- expressions in JavaScript come in four types
 - assignment which assigns a value to a variable
 - arithmetic evaluates to a number
 - string evaluates to a string
 - logical evaluates to a boolean value (true or false)
- use the keyword `var` to declare a variable and optionally assign it an initial value
- a variable declared using `var` with no initial value has the value `undefined`
- it's possible to drop the `var` keyword but that makes the variable global scope -- not recommended

59

JavaScript - assignment

```
var x = 10;
var y = 5;

x += y; // x is now 15 (10 + 5)

x *= y; // x is now 75 (15 * 5)

x /= y; // x is now 15 (75 / 5)

x %= y; // x is now 0 (15 / 5 leaves 0
        //           remainder )
```

60

JavaScript - assignment

```
var x = 10;
var y = 5;
var z;

x++; // increment operator; x is now 11
y--; // decrement operator; y is now 4

z = ++y; // z is 5 and y is now 5 (avoid this)
z = x--; // z is 11 and x is now 10 (avoid too)
```

61

JavaScript - comparison

- use double equals sign (no space) `==` to test if two expressions are equivalent in value
`1 == 1` `"1" == 1` `"100" == 99 + 1`
- use "bang equals" `!=` for not-equal test
`"a" != "A"` `100 != 99.9` `null != undefined`
- comparison operators `<` `>` `<=` `>=` test for less than, greater than, less than or equal, greater than or equal – these 3 are true:
`100 < 111` `"12" < "2"` `"apple" > "Apple"`

62

JavaScript - comparison

- triple equals sign `===` tests if two expressions are equivalent in **value** and the same **type**
`1 === 1` `"cat" === "cat"` `1 === "1"` (false)
- the `!==` tests if two expressions are not equivalent and the same type
`1 !== "1"` (but `1 == "1"` is true)
- `null === undefined` is false but
 `null == undefined` is true

63

JavaScript - logical

- logical AND operator is two ampersands: `&&`
- logical OR operator is two vertical pipes: `||`
- logical NOT operator is a single bang: `!`

```
var x = 10;
var y = 5;

var a = ( x < y ) && ( x == 5 ); // false
var b = ( x > y ) || ( x < 5 );  // true

var c = !b; // c is false
```

64

JavaScript - conditional

- ternary operator as in C, C++

`(expression) ? value1 : value2;`

– if *(expression)* evaluates **true**, then **value1** is returned; otherwise, **value2** is returned

```
var a = ( 3 == 4 ) ? "y" : "n";    // a is "n"
```

- can lead to cryptic programming code if overused

65

Bit Manipulation operators

- JavaScript operators shift bit representations

Operator	Name	Description
&	Bitwise AND	Performs an AND on each bit position
	Bitwise OR	Performs an OR on each bit position
^	Bitwise XOR	Set result bit to 1 only if either not both bits is 1
~	Bitwise NOT	Inverts the bits of the operand
<<	Bitwise left shift	Shift all the bits to the left, leftmost bit dropped
>>	Bitwise right shift	Shift all the bits to the right, keep the sign
>>>	Bitwise zero-fill right shift	Shifts all the bits to the right

66

Bit Manipulation examples

Expression	Result	Binary Description
15 & 9	9	1111 & 1001 = 1001
15 9	15	1111 1001 = 1111
15 ^ 9	6	1111 ^ 1001 = 0110
~15	0	~1111 = 0000
9 << 2	36	1001 shifted 2 bits left = 100100
9 >> 2	2	1001 shifted 2 bits right = 10
19 >>> 2	4	10011 shifted 2 bits right = 100

67

Order of Precedence

Precedence	Operator
1	. (dot operator) [] new
2	Function call ()
3	++ --
4	! (logical not) ~ (bitwise not) + (unary) - (unary) typeof void delete
5	* / %
6	+ (addition) - (subtraction)
7	<< >> >>>
8	< <= > >= in instanceof
9	== != === !==
10	& (bitwise and)
11	^ (bitwise xor)
12	(bitwise or)
13	&& (logical and)
14	(logical or)
15	? :
16	yield
17	= += -= *= /= %= <<= >>= >>>= &= ^= =
18	, (comma operator)

68

JavaScript – if block

- test a condition is met with an if else block

```
if ( expression ) {  
    block of statement(s) to execute if  
                                expression true  
}
```

// do not forget the matching closing brace }

```
var diff = 3-2;  
if (diff == 1) {  
    document.writeln("diff is 1");  
}
```

69

JavaScript – if block

if-else statement version:

```
if ( expression ) {  
    block of statements if expression true  
} else {  
    block of statements if expression false  
}
```

```
var diff = 3-2;  
if (diff == 1) {  
    document.writeln("diff is 1");  
} else {  
    document.writeln("diff is NOT 1");  
}
```

70

JavaScript - if block

```
var day = "Sunday";  
  
if ( day == "Saturday" ) {  
    document.writeln("It's the weekend!");  
    the_weekend = true;  
} else {  
    document.writeln("Back to work.");  
    the_weekend = false;  
}
```

71

JavaScript – if block

- multiple tests combined into one if statement

```
var day = "Sunday";  
var message;  
if ( day == "Saturday" ) {  
    message = "It's the weekend!";  
} else if ( day == "Monday" ) {  
    message = " Back to work. ";  
} else if ( day == "Friday" ) {  
    message = " TGIF ! ";  
} else {  
    message = " Just another day. ";  
}
```

72

JavaScript – if block

- when statement blocks are just one statement, the { } braces are optional

```
var day = "Sunday";
var message;
if ( day == "Saturday" )
    message = "It's the weekend!";
else if ( day == "Monday" )
    message = " Back to work. ";
else if ( day == "Friday" )
    message = " TGIF ! ";
else
    message = " Just another day. ";
```

73

Nested if blocks

- it is possible to nest if statements within another if statement

```
var x = (2-3);
if (x < 0)
    sign = -1;
else {
    if (x == 0)
        sign = 0;
    else
        sign = 1;
}
// acceptable form
```

```
var x = (2-3);
if (x < 0)
    sign = -1;
if (x == 0)
    sign = 0;
if (x > 0)
    sign = 1;

// not recommended form
```

74

JavaScript - switch

- JavaScript switch statement tests an expression against a list of values

```
switch ( expression ) {
    case value1 :
        statement(s)
        break;
    case value2 :
        statement(s)
        break;
    ....
    default :
        statement(s)
}
```

if *expression* matches *value1*, then do these statements only.

if *expression* does not find a match, then default applies.

75

JavaScript - switch

- JavaScript switch is similar to if-else statement

```
if (expression == value1) {
    statement(s) for value1
} else if (expression == value2) {
    statement(s) for value2
} else {
    statement(s) for the default
}
```

76

JavaScript - switch

- JavaScript switch statement tests an expression against a list of literal or expression values

```
var day = "Sunday";
switch ( day ) {
  case "Saturday" :
    document.write("Weekend started.");
    break;
  case "Monday" :
    document.write("Back to work.");
    break;
  default :
    document.write("Another day.");
    break;
}
```

String literals

77

```
<script type="text/javascript">
var name = prompt("Enter your name:",
                  "visitor");

document.write("Welcome " + name );
var sign = prompt("What is your zodiac sign?");

switch(sign.toLowerCase() ) {
  case "aries" :
  case "taurus":
  case "gemini":
    document.write("You are witty and smart.");
    break;
  case "virgo":
  case "capricorn":
  case "libra":
    document.write("You are cool and hip.");
    break;
  default:
    document.write("You are fun and adventurous.");
    break;
}

</script>
```

78

JavaScript - confirm

- confirm method allows the user to select an OK button or a Cancel button
- confirm returns true if OK clicked, false if Cancel clicked

```
if (confirm("Press OK to retry.))
  response = prompt("What is 2+2 ?", "3");
```

79

JavaScript - sample 2

- mathtest.html demonstrates the JavaScript confirm method in action

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

// define variables

var question = "What is 10 + 10?";
var answer = 20;
var correct = '';
var incorrect = '';
```

1 of 3

80

```
// ask the question

var response = prompt(question, "0");

// check the answer

if (response != answer) {

    // wrong answer; retry once more.

    if (confirm("Wrong! \
        Press OK for a second chance."))

        response = prompt(question, "0");
}
```

2 of 3

81

```
// Check the answer.

var output =
    (response == answer) ? correct : incorrect;

// output will be one of these two strings:
    ''
    ''
</script>
</head>

<body>
<script type="text/javascript">

    document.write(output);

</script>
```

3 of 3

82

JavaScript – object literal

- a JavaScript object literal is delimited by { } which contains the object's *properties* as name:value pairs, separated by commas.

```
var student = { name: "Smith, John",
                id: 103923,
                program: "CSC",
                dob: new Date(1990, 3, 20) };

document.write( student.name ); // Smith, John
document.write( student["program"]); // CSC
```

83

JavaScript - eval

- `eval()` method
 - evaluates a string parameter to its numeric value
 - e.g. `eval("4 + 5")` returns a value of 9
 - avoid using eval if possible – there are potential side effects, especially if the string parameter contains malicious code
- <http://javascriptweblog.wordpress.com/2010/04/19/how-evil-is-eval/>

84

JavaScript - iteration

- iteration is the process of repeating the execution of one or more statements until some end condition is reached
 - each time the iteration body is executed is a *cycle*
- example 1 : continually prompt user until right answer is entered
- example 2 : display the month names (January, February, etc) of the entire year
- example 3 : calculate and show the values of a multiplication table up to 12 x 12

85

JavaScript - iteration

- the while statement indicates iteration
- conceptually:
 while (condition is true)
 perform these statement(s) within
 body of iteration in order continually
- in practice:
 while (expression) {
 one or more statements;
 }

86

JavaScript - iteration

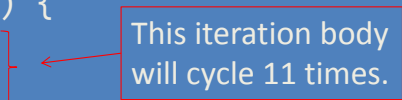
- the expression must evaluate true for the statements in the iteration body to be executed
- implies it is possible for the iteration body to be not executed at all if the expression is false initially

87

JavaScript - iteration

```
var a = 0;
var sum = 0;
while (a <= 10) {
    sum += a;
    a++;
}
document.writeln("sum of 1 to 10 = " + sum);

var answer = 0;
while (answer != 10) {
    answer = prompt("What is 5 + 5?", "0");
}
```



88

JavaScript - iteration

- some gotcha's using while
 - no semi-colon allowed between the condition and iteration body – this leads to a never-ending loop (aka infinite loop)

```
while ( a < 10 ) ; { // oops, an infinite loop !
  a++;
}
```
 - condition must at some point become false
 - braces may be omitted if iteration body is one statement

```
while ( a < 10 ) a++;
```

89

JavaScript - iteration

- some gotcha's using while
 - sometimes while condition is always true but within the iteration body there is a break to end the loop

```
while (true) {
  ... if ( some condition ) break;
}
```
 - condition expression can be an assignment statement by mistake -- watch the equals sign!

```
while ( a = 0 ) vs while ( a == 0 ) // first is false
while ( a = 1 ) vs while ( a == 1 ) // first is true
```

90

JavaScript - iteration

- Some gotcha's using while
 - forgetting to increment the counter if it is used in the condition

```
var n = 0;
var sum = 0;
while ( n < 10 ) {
  sum += n; // oops, n is always zero
}
```

91

JavaScript - iteration

- another form of iteration: for
- useful when number of iterations is known
- conceptually:
 - for (each step in loop counter)
 - execute statement(s)
- in practice:
 - for (optional initial statement(s);
 - condition;
 - optional end body statement(s))
 - execute statement(s)

92

JavaScript - iteration

```
var sum = 0;
for (var n = 0; n <= 10; n++) {
    sum += n;
}
document.writeln("sum of 1 to 10 = " + sum);

var pets = new Array( "cat", "dog", "fish" );
for (var i=0, len=pets.length; i < len; i++) {
    document.write("I have a "
        + pets[i]
        + ". <br />");
}
```

93

JavaScript - iteration

- if only one statement in body, braces may be omitted

for (var n = 0; n < 10; n++) sum += n;
- the initial statement and end statement (usually an increment) are optional
var n = 0;
for (; n < 10 ;) {
 ... n++;
}
- same as a while loop

94

JavaScript - iteration

- if the condition is false initially, the iteration body will not be executed at all and execution will proceed with the next statement after the end of the iteration body

```
var sum = 0;
for (var n = 0; n > 1; n-- ) { // oops, 0 should be 10
    sum += n; // never executed
}
document.write( "sum is " + sum ); // sum is 0
```

95

JavaScript - iteration

- the do while iteration is similar to while but the condition is after the iteration body
- guarantees the iteration body is executed at least once

```
var n = 0;
do {
    n++;
    document.write("n has value "
        + n );
} while ( n < 10 );
```

96

JavaScript - iteration

- an iteration body may include an iteration
- "outer loop" contains an "inner loop"

```
var a = 0;
while ( a < 10) {
    var b = 0;
    while ( b < 10) {
        document.writeln( a * b );
        b++;
    }
    document.writeln( "< /br>" );
    a++;
}
```

97

JavaScript - iteration

```
for (var a = 0; a < 10; a++ ) {
    for ( var b = 0; b < 10; b++ )
        document.writeln( a * b );
    document.writeln("<br />");
}
```

98

JavaScript - iteration

- labels are used to assign a unique identifier to a location within the JavaScript code
 - usage is label_name followed by a colon at the start of a line (after any white space is removed)
- label names cannot be JavaScript reserved words, case-sensitive rule applies!

```
label_one :
    var a = 0;
    while ( a < 10 ) { ...
```

99

JavaScript - iteration

- the break statement terminates the innermost while, do while, for, or switch immediately and transfers control to the following statement
- the break *label* form terminates the specified enclosing label statement

```
var n;
for (n = 0; n < 10; n++ ) {
    if ( n == 5 )
        break;           // immediately exit for loop
} // n is 5
```

100

JavaScript - iteration

- another example of the break in an iteration

```
while (true) {  
    ... continuously process some steps  
    ... if ( a condition becomes true )  
        break;  
}
```

101

JavaScript - iteration

- the continue statement immediately causes the iteration body to start at the next cycle
 - subsequent statements in the iteration body are not executed in the current cycle
 - execution begins at the start of the iteration body (while loop) or with the counter increment (for loop)
 - continue may be used only within the for or while loop

102

JavaScript - iteration

```
// Sum up the odd integers from 0 to 20.  
var sum = 0;  
  
for ( var a=0; a <= 20; a++) {  
    if ( a % 2 == 0 ) {  
        continue;  
    }  
    sum += a;  
}
```

103

JavaScript - iteration

- break and continue may indicate an optional label, e.g.
 - break calculateSum;
 - continue releaseMemory;
- break *label* means stop executing the statement at label (likely a loop of some kind)
- continue *label* means transfer execution to the statement at label

104

JavaScript - iteration

```
Outer:
for ( var a=1; a <= 10; a++ ) {

Inner:
  for ( var b=1; b <= 10; b++ ) {

    document.write( (a*b) + "  " );
    if ( a > 5 ) {
      break Inner;
    }
  }
  document.write( "<br />" );
}
```

```
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
6
7
8
9
10
```

105

JavaScript - iteration

```
Outer:
for ( var a=1; a <= 5; a++ ) {

Inner:
  for ( var b=1; b <= 5; b++ ) {
    if ( a > 5 ) {
      continue Inner;
    }
    document.write( (a*b) + "  " );
  }
  document.write( "<br />" );
}
```

```
1 2 3 4 5
2 4 6 8 10
3 6 9 12 15
```

106

JavaScript - iteration

- use the while iteration when you do not know in advance the number of iterations
- use the for iteration when you do know in advance the number of iterations
- avoid use of break and continue if possible
 - misuse or overuse can lead to 'code spaghetti'



107

JavaScript - function

- the function definition statement consists of the **function** keyword, followed by:
 - the name of the function
 - a list of arguments enclosed in parenthesis and separated by commas
 - the JavaScript statements that define the function, enclosed by braces { }

108

JavaScript - function

- the function declaration defines a set of statements which performs a task

```
function function_name(parameter(s)) {  
    statement block  
}  
e.g.  
function printName( myname ) {  
    document.write("Your name is <b>");  
    document.write( myname );  
    document.write("</b>");  
}  
printName( "Clark Kent" ); // Your name is <b>Clark Kent</b>
```

109

JavaScript - function

- JavaScript functions are usually defined in the header element in the HTML or in separate file
- this ensures that all functions have been parsed before it is possible for user events to invoke a function
- function name rules same as for variables
 - if you accidentally name a variable having the same function name, the variable overrides the function
- parameter names are separated by commas
- no type checking is performed on arguments

110

JavaScript - function

- a function may return a value

```
function calculateArea(height, width) {  
    return height * width; // returns a number  
}  
  
var h = 100;  
var w = 25;  
var area1 = calculateArea(h, w); // area1 is 2500  
var area2 = calculateArea(h, 33); // area2 is 3300
```

111

JavaScript - function

- a function may return a string

```
function encode(message) {  
  
    var coded = "";  
    for (var i=0,len=message.length; i<len; i++) {  
        var ch = message[i];  
        if ( /[a-z]/i.test(ch) ) { // is the character a-z ?  
            coded +=  
                String.fromCharCode( ch.charCodeAt(0) + 1);  
        } else {  
            coded += ch;  
        }  
    }  
    return coded;  
}  
  
var m = "April is a happy month!";  
var x = encode(m); // x is "Bqsjm jt b ibqqz npoui!"
```

112

JavaScript - function

- **primitive** parameters (strings, numbers) are **passed by value**, meaning if the function changes the parameter values, the change is **lost** when the function returns or ends

```
function calculateArea(height, width) {  
    height += 10;  
    return height * width;  
}  
  
var h = 100;  
var w = 25;  
var area1 = calculateArea(h, w); // area1 is 2750  
document.write( h ); // h is 100
```

113

JavaScript - function

- **non-primitive** parameters (arrays, objects) are **passed by reference**, meaning if the function changes the parameter **properties**, the change is **kept** when the function returns or ends

```
function foo(a, obj) {  
    a[2] += 10;  
    obj.name = "parsnip";  
    obj = { name: "carrot"; } // assign new object works only within function  
}  
  
var arr = [1, 2, 3];  
var w = { name: "turnip" };  
foo(arr, w);  
document.write( arr + " " + w.name ); // arr is [1,2,13], w.name is parsnip
```

114

JavaScript functions

- in JavaScript functions are first-class objects
 - can be manipulated and treated like objects
- keyword **Function** defines a function object dynamically created at run-time
 - new `Function(optional param1, param2, ..., body of function as a string)`;
var fun = new Function(a, "return a");
var g = fun();
- sample HTML file testing.html

115

```
<script type="text/javascript">  
  
    // define function testQuestion()  
  
    function testQuestion(question) {  
  
        // define local variables  
  
        var ftmp = new Function(' return ' + question);  
  
        var answer = ftmp(); // answer is 9  
        var output = "What is " + question + "?";  
        var correct = '';  
        var incorrect = '';  
    }  
</script>
```

116

```
// ask the question
```

2 of 3

```
var response = prompt(output, "0");
```

```
// check the result
```

```
return (response == answer) ?  
        correct :  
        incorrect;  
}
```

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

117

```
<body>
```

3 of 3

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

```
var result = testQuestion("4 + 5");
```

```
document.write(result);
```

```
</script>
```

118

JavaScript functions

- functions may be defined inside within a function
 - inner function is private to outer function
 - inner function can be accessed only from the outer function
 - inner function can use arguments and variables of outer function but outer cannot use inner's arguments or variables

119

JavaScript - functions

```
function foo(c) {  
    var x = 100;  
  
    function bar(arg1, arg2) {  
        if (x > 99) arg1++; // x access allowed in inner function  
        return (arg1 + arg2);  
    }  
    return bar(x, c); // returns 101 + 10  
}  
  
var n = 10;  
var p = foo(n); // p is 111 (101 + 10)  
document.write(p);  
document.write( bar(10,10) ); // not allowed - out of scope
```

120

JavaScript – recursive function

- functions may be recursive; a function may call itself inside the function declaration

```
function factorial(n) {  
    if ((n == 0) || (n == 1))  
        return 1;  
    else  
        return (n * factorial(n - 1));  
}  
  
var a = factorial(4); // a gets value 24
```

121

JavaScript – function arguments

- arguments of a function are kept in an array-like object named **arguments**

```
function sumup(n) {  
    var sum = 0;  
    for (var i = 0; i < arguments.length; i++) {  
        sum += arguments[i];  
    }  
    return sum;  
}  
  
var a = sumup(3,4,5); // a gets value 12  
var b = sumup(1,-3,1,3,4); // b gets value 6
```

122

JavaScript – Number and String

- JavaScript has built-in functions Number and String to convert an object to that type

```
var today = new Date;  
var x = String( today );  
    // x is "Mon Aug 20 04:37:33 GMT-0700 2012"  
  
var str = "123";  
var n = Number( str ); // n is 123
```

123

JavaScript – exception

- handling potential errors during run time is important
- the **throw** statement provides error handling
- in JavaScript any object can be thrown, though it is usually a number or a string

```
Examples:  
throw "Error 100";  
throw 1033;  
throw false;  
throw ReferenceError();
```

124

JavaScript – try catch

- the **try...catch** marks a block of statements to try, and if there is an error (or 'exception'), the catch block controls the process neatly
- An exception 'thrown' within a **try** block is managed in the 'catch' block

```
var a = 0;
try {
    if (a == 0) throw "a is zero";
}
document.write("a not zero"); // never executed because exception is thrown.
} catch (e) {
    // e is the exception parameter.
    // a is found to be zero
    if (e == "a is zero")
        document.write("error - a is zero");
}
```

125

JavaScript – finally

- with the try...catch blocks is an optional **finally** block
- code within the **finally** block will execute after the try and catch blocks execute but before the statements following the try...catch
- **finally** block executes whether or not an exception is thrown
- use **finally** block to release a resource your code is using (such as an open file)

126

JavaScript – finally


```
try {
    openMyFile();
    writeSomeData(theData); // May cause error.
} catch (e) {
    handleError(e); // If error, handle it.
} finally {
    closeTheFile(); // Always close file.
}
```

127

JavaScript – Error object

- the **Error object** in JavaScript allows you to create your own error message object and throw it
- IE browser supports an optional **number**
- Firefox browser supports a **message**, filename, line#
- Opera, Chrome, Safari support only **message**

```
var a = 0;
try {
    if (a == 0) throw new Error("a is zero");
}
document.write("a not zero"); // never executed because exception is thrown.
} catch (e) {
    // e is the exception parameter.
    // a is found to be zero
    if (e.message == "a is zero")
        document.write("error - a is zero");
}
```



128

JavaScript – variable scope

- when you declare a variable outside of any function, it is called a *global* variable because it is visible to any other JavaScript code in the current document
- if you declare a variable inside a function, it is *local* to that function only and not visible to JavaScript outside that function
- if the function declares a new local variable having the same name as a global, the function uses the local variable

129

JavaScript – scope ex 1

```
var a = 100;    // global

function myfun() {
    var b = 200;    // local to myfun only
    a++;            // variable a is global
    var c = b + a;  // variable c is local
    document.write( c );
}

myfun(); // call function myfun, displays 301
document.write( a ); // displays 101
if ( typeof c !== undefined )
    document.write( c ); // nothing displayed
```

130

JavaScript – scope ex 2

```
var a = 100;    // global

function myfun() {
    var b = 200;    // local to myfun only
    var a = 10;     // local - not the global
    a++;            // this variable a is local
    var c = b + a;  // variable c is local
    document.write( c );
}

myfun(); // call function myfun, displays 211
document.write( a ); // displays 100
if ( typeof c !== undefined )
    document.write( c ); // nothing displayed
```

131

JavaScript – scope ex 3

```
var a = 100;    // global

function myfun() {
    var b = 200;    // local to myfun only
    var a = 10;     // local - not the global
    window.a++;     // this variable a is global
    var c = b + window.a; // variable c is local
    document.write( c );
}


myfun(); // call function myfun, displays 301
document.write( a ); // displays 101
if ( typeof c !== undefined )
    document.write( c ); // nothing displayed
```

132

JavaScript - scope

- JavaScript uses *hoisting* to move the declaration of any declared variables within a function to the *top* of the function

```
function myfun1() {  
  document.write( a + b );  
  var a = 10;  
  var b = 20;  
}  
function myfun2() {  
  var a, b;  
  document.write( a + b );  
  a = 10, b = 20;  
}
```



Identical functions

133

JavaScript - object

- objects in JavaScripts are similar to objects in real life with properties, type, and behaviour
- A **car** object has **properties**:
 - colour, make, model, year, VIN, transmission, manufacturer
- A **car** object has **type**:
 - car is a type of a vehicle
- A **car** object has **behaviour**:
 - accelerate, decelerate, turn left, turn right, stop

134

JavaScript Date object

- a Date object in JavaScript represents a single date
- three different usages:

```
variable = new Date( parameters );
```

where the parameters indicate year, month, day, hour, minute, second, milliseconds in order
 - if no parameters, current date assumed; otherwise year, month and day must be provided
 - if hour and minute not provided, then midnight assumed (0 hour, 0 minute)
 - if year < 100, then 1900 + year is assumed

135

JavaScript Date object

```
variable = new Date("date string" );  
where the date string represents a text form of  
the date as in  
"October 7, 1995"  
"October 7, 1995 12:43"
```



```
variable = new Date(milliseconds);  
where milliseconds is an integer value  
representing the number of milliseconds since  
1 January 1970 00:00:00 UTC (Unix Epoch)
```



```
new Date(1343053807040);
```

136

JavaScript Date object

- UTC (Universal Time coordinated) is a timezone-independent method of storing time values, based on milliseconds since midnight, January 1, 1970 in the Greenwich Mean Time zone
- all dates and times are stored internally in JavaScript using UTC format
- Date objects have both UTC and non-UTC methods to get and set date and time values
- https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/Date

137

JavaScript Date object

```
var today = new Date();
var birthday = new Date( 1962, 7, 24); // Aug 24, 1962
var party = new Date( 96, 3, 23, 8, 0, 0);
// Apr 23, 1996, 8:00AM
var d1 = new Date(2012, 4, 12); // May 12, 2012

var d2 = new Date("November 3, 2011");
var d3 = new Date("May 1, 2011 9:00 PST");

var d4 = new Date(1343053807040);
// July 23, 2012 3:33 PM PST
```

138

JavaScript Date object

- if the Date cannot be determined to be valid, the Date is set to be "Invalid Date"
- if the new keyword is not used to create the Date object, then the date value is returned as a string object rather than a Date object
- Date objects can be subtracted from each other to obtain the amount of separation time in milliseconds

139

JavaScript Date object

```
var today = new Date(); // current date and time

var yesterday = new Date(2012, 7, 23);

var elapsed = today - yesterday;
// number of millisecs since start of Aug 23, 2012 (00:00)

elapsed = elapsed / (60 * 60 * 24 * 1000);
// number of hours since start of Aug 23, 2012 (00:00)
```

140

JavaScript Date object

- Date object methods:
 getDate() returns the day of the month (1-31)
 getFullYear() returns the year in four digits
 getMonth() returns the month (0 – 11)
 getTime() returns milliseconds since midnight
 Jan 1, 1970
 plus many more methods ... check link
 https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Date/

141

JavaScript Date object

```
var today = new Date("May 2, 2012 5:15 PM");
var yr    = today.getFullYear();
var month = today.getMonth();
var day   = today.getDate();
var hr    = today.getHours();
var min   = today.getMinutes();
document.write( "Today is "
               + yr + " "
               + month + " "
               + day + " "
               + hr + " "
               + min );

// displays: Today is 2012 4 2 17 15
```

142

JavaScript Date object

- third party JavaScript libraries available for parsing, manipulating, and formatting dates
 - Date.js
<http://www.datejs.com/>
 - Moment.js
<http://momentjs.com/>
 - dateFormat.js
<http://blog.stevenlevithan.com/archives/date-time-format>
 - Date Extensions
http://depressedpress.com/javascript-extensions/dp_dateextensions/

143

JavaScript - array

- an array literal is a list of zero or more expressions, each an array element, enclosed by square brackets []
- the length of the array literal is the number of elements it contains
- array elements are referenced by [index]

```
var pets = [ "cat", "dog", "fish" ]; // array pets
document.write (pets.length); // displays 3
document.write( pets[0] );    // displays cat
document.write( pets[5] );    // displays undefined
```

144

JavaScript - array

```
var pets = [ "cat", "dog", "fish" ];  
  
for (var i=0, len=pets.length; i < len; i++) {  
    document.write("I have a pet "  
        + pets[i]  
        + "<br />");  
}  
  
// use a for loop to iterate over the values of an array
```

145

JavaScript - array

- array elements need not be all the same primitive data type
`var myList = ["cat", 1000, false, (1==2-1)];`
- array elements may contain variables
`var a = -333.33;
var myList2 = ["dog", a, 100];`
- array elements may be literal arrays as well
`var myList3 = [[1,2], ["cat", "mouse"], 0.01];
var myList4 = ["fish", myList];`
but the array element counts as a single

146

JavaScript – array literal

- in JavaScript you can omit specifying all the elements in an array literal
`var zoo = ["tiger", , "bear", , "lion"];`
has 5 array elements – the second and fourth elements are undefined
- declaring an array with no initial elements
`var emptyList = [];`

147

JavaScript – array object

- array objects are the same as array literals only defined differently using the `Array` keyword
- no difference but literal format is shorter

```
var a_obj = new Array( 1, 2, 3 );  
var b_obj = Array(300, 301, 302);
```

148

JavaScript – array - adding

- adding new elements to an array is a simple matter of assigning them based on a new index

```
var a = [];           // array a is empty
a[1] = "cat";
                        // index 0 element is undefined
a[3] = "dog";
                        // array a is length 4 but elements at
                        // index 0 and 2 are undefined.
```

149

JavaScript – array - index

- adding new elements to an array using a non-integer index causes a new property for the array, instead of an array element

```
var a = [];           // array a is empty
a[1.5] = "clip";      // legal, but no element
if ( a.hasOwnProperty[1.5] ) {
    document.write("property is set");
}
```

150

JavaScript – array - splice

- removing an element from an array requires the `splice` function

array_name.splice(index, number of elements)

```
var a = [ "cat", "and", "dog" ];
a.splice( 1, 1 );    // a is [ "cat", "dog" ]
```

151

JavaScript – array - delete

- the `delete` keyword can be used to swap an array element value with `undefined`
- using `delete` in this way does not remove the element itself or shorten the array

```
var a = ["sun", "moon", "earth"];
delete a[1];
        // a is "sun", , "earth"
        // length of a is still 3
```

152

JavaScript – array - push

- another way to add new elements to an array in JavaScript is to use the array's push function
- elements are always added to the end

```
var a = ["cat", "and"];  
a.push("the");  
a.push("dog", "story");  
    // array a now has 5 elements
```

153

JavaScript – array - pop

- pop removes the last element in an array and returns it – if the array is empty, undefined is returned.

```
var a = ["cat", "and", "dog"];  
var b = a.pop();  
    // array a is ["cat", "and"]  
    // b is "dog"
```

154

JavaScript – array - reverse

- the reverse method moves all the elements in the array into reverse order

```
var batman = ["West", "Keaton", "Kilmer",  
             "Clooney", "Bale"];  
batman.reverse();  
document.write( batman[0] ); // Bale
```

155

JavaScript – array - foreach

- the foreach method defines a call back function to be applied to each element in the array
- array element values cannot be changed this way

```
var sum = 0;  
function sumthis(value) {  
    sum += value;  
}  
var a = [ 111, 22.2 ];  
a.forEach( sumthis ); // sum is 133.2
```

156

JavaScript – array - sort

- the `sort` method moves all the elements in the array into alphabetic order ("30" appears before "2") – use a function for numeric sort

```
var villain = ["Joker", "Catwoman", "Two-Face",  
              "Bane", "Riddler"];  
villain.sort();  
document.write( villain[0] ); // Bane  
  
var s = [ 23, 15, 8, 42, 16, 4 ];  
s.sort( function(a,b) {return a-b});  
// array score is now 4,8,15,16,23,42
```

157

JavaScript – array - join

- the `join` method causes all the array elements to be merged into a single string with a delimiter (comma is the default)

```
var dessert = ["pie", "cake", "sundae"];  
var s = dessert.join();  
// s is "pie,cake,sundae"  
  
var t = dessert.join(" / ");  
// t is pie / cake / sundae
```

158

JavaScript – Regular Expression

- a regular expression describes a string pattern
 - e.g. apply the pattern `/at/` to the string "Cat in the Hat" matches "Cat" in the Hat
 - patterns `/AT/`, `/ta/`, and `/cat/` will find no matches
- metacharacters such as `*`, `+` and `?` are called **qualifiers** and are used in the pattern after a character
 - `*` denotes zero or more matches
 - `+` denotes 1 or more matches
 - `?` denotes either 0 or 1 match

159

JavaScript – Regular Expression

- `/fe*/` matches "fee" in "two feet" and "f" in "left arm" but nothing in "my head"
- `/to+/` matches "to" in "nine toes" and "too" in "me too" but nothing in "my tasks"
- `/h?ea?/` matches "hea" in "my head" and "e" in "left foot" and "ea" in "my ear"

160

JavaScript – Regular Expression

- metacharacter `.` (decimal point) matches any single character except the newline
 - `/r.t/` matches "rat", "rut", "r t" but not "art"
- `\` is used to match metacharacters
 - `/a*/` matches "a*" but not "apple"
- `^` matches beginning of input
 - `/^A/` matches "A story" but not "the ABCs"
- `$` matches end of input
 - `/x$/` matches "the ox" but not "my axe"

161

JavaScript – Regular Expression

- `|` (pipe) matches text on either side
 - `/a|b|c/` matches either the first "a", "b", or "c", and `/apple|pear/` matches either "apple" or "pear"
- `{n}` where n is a positive integer, matches n occurrences of the preceding character
 - `/e{2}/` matches the "ee" in "feed" and the first "ee" in "feeed", but not "fed"
- `{n,m}` where n and m are positive integers, matches at least n and at most m occurrences of the preceding character
 - `/r{1,3}/` matches the "r" in "art", the "rr" in "array", and the first "rrr" in "arrrrh!"

162

JavaScript – Regular Expression

- `[abc]` defines a range of any characters to match. Shorthand range form can use a hyphen `[a-c] = [abc]`
 - `/[a-m]/` matches the "e" in "A pear" and `/[a-z]+/` matches "anana" in "Banana"
 - `/[0-9]/` matches the "4" in "robin4nest"
- negation of the range uses the `^`
 - `/^[a-m]/` matches the "p" in "pear"
 - `/^a-z/` is same as `/^[a-z]/`

163

JavaScript – Regular Expression

- Special characters used in regex
- `\d` matches a single digit – same as `[0-9]`
- `\n` matches a new line
- `\s` matches a single white space, tab, form feed, new line
- `\t` matches a tab
- `\w` matches any alphanumeric including the underscore – same as `[A-Za-z0-9_]`
- `\xHH` matches the character with the hex code HH e.g. `/x20/ = /\s/`

164

JavaScript – Regular Expression

- `\D` matches any non-digit, same as `[^0-9]`
- `\S` matches any non white space
- `\W` matches any non alphanumeric, same as `[^A-Za-z0-9_]`
- `\b` matches a word boundary - `\W\w` or `\w\W`
 - `/\bsp/` matches the "spo" in "my spoon" and no match in "dispose"
 - `/\ba\b/` matches the second "a" in "at a mall"
- `\B` matches a non-word boundary
 - `/\B../` matches the "ec" in "pecan" but not " a"

165

JavaScript – Regular Expression

- `(tree)` matches "tree" and remembers the match using the resulting array's elements `[1],..., [n]`
 - `/([A-Za-z]+\s(\w+))/` matches "John Smith" in "100 John Smith 203-300" and remembers "John" and "Smith" in the resulting arrays `[1]` and `[2]`
 - `([A-Za-z]+)` means look for one or more alphabetic characters (any case) and remember them ... e.g. "John"
 - `\s` matches a single white space
 - `(\w+)` is the same as `([A-Za-z0-9_]+)`

166

JavaScript – Regular Expression

- pattern flags – regular expressions have four optional flags, used singly or combined in any order
 - `g` – indicates global search
 - `/w\s/g` returns "e", "i", "o" in "fee fi fo fum"
 - `/w\s/` returns "e"
 - `i` – indicates case insensitive (ignore case)
 - `/abc/i` is the same as `[A-Ca-c]`
 - `m` – indicates multi-line search
 - makes the `^` `$` characters match the start and end of any input line, as opposed to the entire input text
 - `y` – "sticky" search – match starting at current position in the target string – non-standard

167

JavaScript – Regular Expression

- the qualifiers `*` `+` `?` `{ }` are by default, "greedy"
 - matches will take as much as it can find
 - `/a+b+/` matches "aaaabbbb" in "aaaabbbbabc"
- "lazy" matches will stop as soon as minimum found
 - the `?` qualifier appended to `*` `+` `{ }` makes the match lazy not greedy
 - `/a+?b+?/` matches "aaaab" in "aaaabbbbabc"

168

Mozilla-specific - let

- JavaScript version 1.7 supports the `let` keyword for Firefox browsers – not yet an ECMA standard (in draft)
- useful when you want to use an existing variable name within a separate code block
- need to specify that you wish to use JavaScript 1.7
`<script type="application/javascript;version=1.7"></script>`

```
var x = 10, y = 2;
let (x=5) {
    y = x; // y is now 5
}
document.write(x + " " + y); // 10 5
```

169

Course Note References

- <http://www.ecma-international.org/publications/standards/Ecma-262.htm>
- http://www.reddit.com/r/javascript/comments/fqht8/references_for_javascript_mastery/
- http://www.w3.org/community/webed/wiki/Main_Page
- <http://code.google.com/edu/submissions/html-css-javascript/>
- <http://reference.sitepoint.com/css>
- <https://developer.mozilla.org/en-US/docs>

170