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 { }

109

JavaScript - function

- JavaScript functions are usually defined in the <head> 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 the 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

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
```


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!"
```

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
```


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
```


JavaScript functions

- In JavaScript functions are first-class objects
  - can be manipulated and treated like objects
- The 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();
```



```
<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 = '<img src="correct.gif">';
   var incorrect ='<img src= "incorrect.gif">';
```

```
// ask the question                                    2 of 3

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

// check the result

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

   </script>
   </head>
```



```
                                                       3 of 3
<body>

   <script type="text/javascript">

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

   document.write(result);

   </script>
```


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

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
```

outer function is foo

inner function is bar

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
```


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
```


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();
```


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

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
```


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
```


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
```


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

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Scope_Cheatsheet

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

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

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);

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

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
```


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

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)
```


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/

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
```


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/

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
```


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
```

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

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 = [];
```


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);
```


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.
```

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");
}
```


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" ]
```


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
```


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
```

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"
```


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
```


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
```


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
```

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 delimiter)

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

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


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

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"

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"

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 "feeeed", 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 "arrrrgh!"

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]/

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/

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
  - /\bspo/ 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"

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_]+)`

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

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"

Mozilla-specific - let

- JavaScript version 1.7 supports the `let` keyword for Firefox browers – 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
```

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