Topics

Operators
Control Structures
Functions
Object Reference
Array Reference
Date Reference
Wrapper References
Arithmetic Operators

*  *= multiplication
/  /= division
%  %= modulus
+  += add (if at least one operator is a string, string concatenation)
-  -= subtract
++  -- prefix, postfix increment/decrement (unary)

\[ x = x + 1; \] is equivalent to \[ x += 1; \]

Boolean and Relational Operators

!  logical NOT
&&  logical AND
||  logical OR
<, <= less than, less than or equal to
>, >= greater than, greater than or equal to
==, !== equal in value, not equal in values
===, !=== Equal in value and type, not equal in value and type

Things that are truthy | Things that are falsey
---|---
Objects | null
Non-empty string | Empty string
Non-zero numbers (including Infinity, -Infinity) | Zero, NaN
Undefined | NaN

http://einstein.etsu.edu/~pittares/CSCI3110/examples/4-1.htm
Bitwise Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>~</td>
<td>bitwise not (flips 0 bits to 1 and vice versa) (unary)</td>
</tr>
<tr>
<td>&amp;</td>
<td>bitwise and (binary)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>bitwise exclusive or (binary)</td>
</tr>
<tr>
<td>&lt;&lt; x</td>
<td>left shift bits x positions, pad with 0.</td>
</tr>
<tr>
<td>&gt;&gt; x</td>
<td>right shift bits x position, maintain sign (positive/negative)</td>
</tr>
<tr>
<td>&gt;&gt;&gt; x</td>
<td>unsigned right shift bits x position, sign bit shifted as well</td>
</tr>
</tbody>
</table>

Control structures

if, while, do/while, for, for/else, switch as in other languages.
No foreach, but instead for...in which enumerates the properties of an object or the elements in an array.
Functions

**Function declarations** begin with reserved word `function` do not specify a return type in the interface do not permit overloading of function name accept a variable number of parameters

- Stored in a 0 based array named **arguments**
  - [http://einstein.etsu.edu/~pittares/CSCI3110/examples/4-2.htm](http://einstein.etsu.edu/~pittares/CSCI3110/examples/4-2.htm)
- Explicitly named arguments that are omitted are undefined
  - [http://einstein.etsu.edu/~pittares/CSCI3110/examples/4-3.htm](http://einstein.etsu.edu/~pittares/CSCI3110/examples/4-3.htm)

Parameters are passed by value

---

Alternate function definition syntax

These two function definitions are roughly equivalent.

**Function declaration**

```javascript
function demo(x, y) {
    return x * y;
}
```

**Function expression**

```javascript
var demo = function(x, y) {
    return x * y;
};
```
Function run-time behavior

All function declarations are loaded into the execution context before any code is executed. This execution context is bound to a single script block.

http://einstein.etsu.edu/~pittares/CSCI3110/examples/4-3a.htm

Function expressions are loaded as flow of execution reaches that line.

http://einstein.etsu.edu/~pittares/CSCI3110/examples/4-3b.htm

Scoping reminder

Variables in JavaScript exist in a "scope chain".

Placing var in front of a variable when declared places it in the immediate scope context.

When a variable is referenced, the immediate scope is checked. If variable is not found, the global scope is checked.

Variables do not go out of scope when their block is terminated.

http://einstein.etsu.edu/~pittares/CSCI3110/examples/4-4.htm
http://einstein.etsu.edu/~pittares/CSCI3110/examples/4-5.htm
http://einstein.etsu.edu/~pittares/CSCI3110/examples/4-6.htm
http://einstein.etsu.edu/~pittares/CSCI3110/examples/4-7.htm

Write your code as if JavaScript scoping were traditional. Remember unusual scoping in debugging.
### Storing values in Object

JavaScript provides the **Object reference** type to allow flexible storage of data.

http://einstein.etsu.edu/~pittares/CSCI3110/examples/4-8.htm

Although dot notation most typical, bracket notation is also accepted for property reference.

http://einstein.etsu.edu/~pittares/CSCI3110/examples/4-9.htm

"Literal notation" is also possible for declaration.

http://einstein.etsu.edu/~pittares/CSCI3110/examples/4-10.htm

Anonymous literal notation may be used as a function call parameter.

http://einstein.etsu.edu/~pittares/CSCI3110/examples/4-11.htm
http://einstein.etsu.edu/~pittares/CSCI3110/examples/4-12.htm
http://einstein.etsu.edu/~pittares/CSCI3110/examples/4-13.htm

### Storing values in Array

JavaScript arrays allow varying data types among cells. Arrays are dynamically resizable.

**Multiple syntax options for declaration:**

```javascript
var names = new Array();
var names = new Array(10);
var names = new Array("Jo", "Kelli", "Bob");
```

In all the above, "new" can be omitted

```javascript
var names = ["Jo", "Kelli", "Bob"];
var names = [];
```
Array length

Arrays maintain the property "length". If reference made to a cell beyond pre-specified length, array automatically grown to new length.

http://einstein.etsu.edu/~pittares/CSCI3110/examples/4-14.htm
http://einstein.etsu.edu/~pittares/CSCI3110/examples/4-15.htm

length is a programmer changeable property.
http://einstein.etsu.edu/~pittares/CSCI3110/examples/4-16.htm

length is useful for adding element to end of array.
http://einstein.etsu.edu/~pittares/CSCI3110/examples/4-17.htm

Array operations

Array has toString(), automatically called when outputting array, that outputs array as comma delimited string.

http://einstein.etsu.edu/~pittares/CSCI3110/examples/4-18.htm

arrayname.push(params)—add 1 or more param to end of array
arrayname.pop()—returns last item in array and decrements arrayname.length
arrayname.unshift(params)—add 1 or more param to beginning of array
Associative Arrays in JavaScript?

JavaScript arrays are intended to be used as numeric arrays. Although syntax can be coalesced into something resembling an associative array. To create an associative array type structure, instead use the Object reference type.

http://ajaxian.com/archives/javascript-associative-arrays-considered-harmful

Quick drill
Date object reference

Instantiation without parameters sets object to current date/time in Unix time code. Can pass in Unix time code to set to another date/time.

Date.parse(str)—converts str to Unix time code. str in one of following formats:
- month/date/year (example 12/14/1968)
- monthname date, year (example December 14, 1968)
- timestamp format (Tue Dec 14 1968 07:22:48 GMT-0700)

If invalid data specified browsers handle inconsistently.

http://einstein.etsu.edu/~pittares/CSCI3110/examples/4-19.htm

Date Operations

<table>
<thead>
<tr>
<th>Methods</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>toDateString()</td>
<td>returns date's day of week, month, day of month, and year</td>
</tr>
<tr>
<td>toTimeString()</td>
<td>returns date's hours, minutes, seconds, and time zone</td>
</tr>
<tr>
<td>getTime(), setTime(UTC)</td>
<td>returns/sets Unix timecode of date</td>
</tr>
<tr>
<td>getFullYear(), setFullYear(year)</td>
<td>return/sets year</td>
</tr>
<tr>
<td>getMonth(), setMonth(mon)</td>
<td>returns/sets month</td>
</tr>
<tr>
<td>getDate(), setDate(date)</td>
<td>returns/sets day of month</td>
</tr>
<tr>
<td>getDay()</td>
<td>returns day of week as number</td>
</tr>
<tr>
<td>getHours(), setHours(hr)</td>
<td>returns/sets hours (0 to 23)</td>
</tr>
<tr>
<td>getMinutes(), setMinutes(min)</td>
<td>returns/sets minutes (0 to 59)</td>
</tr>
<tr>
<td>getSeconds(), setSeconds(sec)</td>
<td>returns/sets seconds(0 to 59)</td>
</tr>
</tbody>
</table>
Primitive type reference wrappers

Each JavaScript primitive type (string, number, etc.) has a corresponding reference type that provides methods for working with the primitive.

http://einstein.etsu.edu/~pittares/CSCI3110/examples/4-20.htm
http://www.w3schools.com/jsref/default.asp

Note: Boolean can be troublesome as objects are regarded as true in logical operations. Therefore a Boolean object set to false is still evaluated as true. Advice: **Don't use Boolean objects.**

Topics

Operators
Control Structures
Functions
Object Reference
Array Reference
Date Reference
Wrapper References