CS306 Web Design Notes
2016–2017 Term 3

Reference Sheet: JavaScript and DOM

Name: _____ Section: _____

Types

Declare a weakly-typed variable: **var** *variable*

Type	Literal(s)
`undefined`	**undefined** *
`boolean`	**true** or **false**
`number`	0, -1, 2.718, **NaN**, etc.
`string`	'*text*' or "*text*" †
`object`	{*variable*: *value*, *var*: *val*, ...} or **null**
`Array`	[*value*, *value*, ...] ‡

*All variables are initially set to **undefined** until a different value is assigned to them.

† Certain characters within *text* must be escaped, *e.g.* \n, \', \", and \\

‡ Subtype of object that can also be created by calling constructor: **new** `Array(`*length*`)`

Casting

Each cast function expects a single argument and returns a value of the type being cast to.

Type	Cast function
`boolean`	`Boolean()`
`number`	`Number()` †
	`parseFloat()`
	`parseInt()` ‡
`string`	`String()`

† Returns `NaN` as soon as it encounters a non-digit character in a `string`. Use `parseFloat()` instead to avoid this behavior.

‡ In addition to casting, is often used on `numbers` to truncate their fractional components.

Conditionals

if(*predicate*) {}
else if(*predicate*) {}
else {}

*predicate*s are of type `boolean`. There may be any number of **else if**s, but at most one **else**.

Operators & coercion

In order of execution (adjust w/ ()s):

Op	Left	Right	Coerce	Result
`.`	`object`	*variable*	✓	any
`[]`	`object`	`string`	✓	any
`++`	*variable*	N/A	✓	`number`*
`--`	*variable*	N/A	✓	`number`*
`!`	N/A	`boolean`	✓	`boolean`
`**`	`number`	`number`	✓	`number`
`*`	`number`	`number`	✓	`number`
`/`	`number`	`number`	✓	`number`
`%`	`number`	`number`	✓	`number`
`+`	?	?	†	input
`-`	`number`	`number`	✓	`number`
`<`	any	any	‡	`boolean`
`==`	any	any	∧	`boolean`
`===`	any	any		`boolean`
`!==`	any	any		`boolean`
`&&`	`boolean`	`boolean`	✓	`boolean`
`||`	`boolean`	`boolean`	✓	`boolean`
`=`	*variable*	any		input*

Note that `+=`, `-=`, `*=`, `/=`, and `%=` are also valid assignment operators.

* Has the side effect of changing *variable*.

† Supports both `numbers` and `strings`, but coercion prefers to convert to `string`.

‡ And `>`, `<=`, `>=`. Coercion prefers `number`.

∧ And `!=`. Coercion prefers `number`. `objects` are tested for being the **same** instance.

Loops

while(*predicate*) {}
for(*initializer*; *predicate*; *increment*) {}

e.g. `for(var count = 1; count <= 5; count++) {}`

or `var count = 0` to start from 0; `count+=2` to count by 2

*predicate*s are of type `boolean`. Both types of loop run the body (everything between the curly braces) as long as *predicate* is `true`, stopping as soon as it becomes `false`. Any and all parts of the **for** loop header may be omitted.

Function definitions & calls

Can be called from HTML handlers, *e.g.* an `onclick` or `onload` attribute on any tag.

Define: `function doNothing() {}`
Call: `doNothing();`

Define: `function showAverage(foo, bar) { alert((foo + bar)/2); }`
Call: `showAverage(1, 2); //shows 1.5`

Define: `function average(foo, bar) { return (foo + bar)/2; }`
Call: `alert(average(1, 2)); //1.5`

Object definitions & use

Constructor: `function Subtype(value) { this.variable = value; }`
Functions:
```
Subtype.prototype = {
  replace:  function(newVal) {
   var temp = this.variable;
   this.variable = newVal;
   return temp;},
  toString:  function() {
   return this.variable+'K';},
  };
```
Construct: `var grand = new Subtype(1);`
Update var.: `grand.variable++; //now 2`
Call func.: `alert(grand.replace(3)); //2`
Convert to `string`: `alert(grand); //3K`
Convert to `number`: `parseFloat(grand)` *
* `Number(grand)` is `NaN` because all conversions call `grand.toString()`, which adds a 'K' here.

Array usage

Create: `var arr = ['foo', 'bar'];`
Access first: `alert(arr[0]);`
Count elements: `alert(arr.length);`
Access last: `alert(arr[arr.length - 1]);`
Replace first: `arr[0] = 'foobar';`
Append: `arr[arr.length] = 'baz';`
Sort all elements: `arr.sort();` *
* Compares `string`s or takes optional comparison function. To sort an array of `number`s, do:

```
arr.sort(function(a, b) { return a - b; });
```

Built-in functions

`isNaN(number)` → `boolean`
`Math.abs(number)` → `number`
`Math.max(number, number)` → `number`
`Math.min(number, number)` → `number`
`Math.random()` → `number`
Returns a fractional number ≥ 0 and < 1
e.g. `parseInt(Math.random()*(`*max* `-` *min* `+ 1)) +` *min*
`Math.sqrt(number)` → `number`

DOM `window` object

Can omit the `window.` when calling these:
`alert(string)` `confirm(string)` → `boolean`
`prompt(string, string)` → `string`
First argument is the message
Second (optional) argument is the default

DOM `document` object

Must call as `document.`*function*`()`:
`getElementsByClassName(string)` → `object` *
Returns an array of all matching elements
`getElementsByTagName(string)` → `object` *
`getElementById(string)` → `object`
`write(string)`
Note that this replaces the contents of the whole page if called from within a function!

`<form name="fn"><input name="in"></form>` is also accessible as `document.fn.in`; this object is an array if multiple tags have this same name. `ById` and `ByClassName` select by the `id` (unique) and `class` (grouping) HTML attributes.
* Can also be applied to any individual element within the `document` hierarchy.

DOM `document` elements

All the above *'d `document` functions, plus:
`classList.add(string)`
`classList.contains(string)` → `boolean`
`classList.remove(string)`
`id` (`string`) `innerHTML` (`string`) `style` (`object`) †
`<input>` tags also support the following:
`checked` (`boolean`) `value` (`string`)
† Contains a variable for each CSS property, *e.g.* `backgroundColor` for `background-color`.