CSE 341 Section Handout #9
JavaScript Cheat Sheet

Types

```
Number  String  Boolean  null  undefined  Object  Function
typeof(expr)              // returns a lowercase type string, e.g. "number", "object"
```

Variables

```
var name = expression;
name = expression;        // assign new value, or create global variable
```

Comments

```
// comment     /* comment */
```

Converting between types

```
parseInt(expression)        // integer
parseFloat(expression)      // real number
!!expression                // boolean
"" + expression             // anything -> string
string.split(delimiter)     // string -> array
array.join(delimiter)       // array -> string
```

Statements (very similar to Java)

```
if (test) {               // any type can be
    statements;           // used as a test;
} else if (test) {        // falsey: 0, 0.0, "",
    statements;           // NaN, null, undefined
} else {
    statements;
}

for (initialization; test; update) {
    statements;
}
```

```
while (test) {
    statements;
}

do {
    statements;
} while (test);

for (var name in obj) {
    statements;   // sucks; don't use
}
```

Functions

```
function name(paramName, paramName, ..., paramName) {
    statements;
}
```

`Math` object

Method	**Description**	**Method**	**Description**
`Math.abs(`*value*`)`	absolute value	`Math.min(`*value1, ...*`)`	smaller of *n* values
`Math.ceil(`*value*`)`	rounds up	`Math.pow(`*base, exponent*`)`	*base* to the *exponent* power
`Math.cos(`*value*`)`	cosine, in radians	`Math.random()`	random real *k* in $0 \le k < 1$
`Math.floor(`*value*`)`	rounds down	`Math.round(`*value*`)`	nearest whole number
`Math.log(`*value*`)`	logarithm, base *e*	`Math.sin(`*value*`)`	sine, in radians
`Math.max(`*value1,...*`)`	larger of *n* values	`Math.sqrt(`*value*`)`	square root

Constant	**Description**
`Math.E`	2.7182818...
`Math.PI`	3.1415926...

Input/output

```
print(expression)
print(expression, expression, ..., expression)
```

CSE 341 Section Handout #9

Relational/Logical Operators			
Operator	**Description**	**Operator**	**Description**
`<`	less than	`===`	equal (strict about types)
`<=`	less than or equal	`!==`	not equal (strict about types)
`>`	greater than	`&&`	and
`>=`	greater or equal	`||`	or
`==`	equal (loose about types)	`!`	not
`!=`	not equal (loose about types)		

Strings

String Method/Property	Description
`contains(`**str**`)`	true if this string contains the other's characters inside it
`charAt(`**index**`)`	character at a given index, as a 1-letter string
`charCodeAt(`**index**`)`	ASCII value at a given index, as an integer
`String.fromCharCode(`**expr**`)`	converts an ASCII code into its character equivalent
`indexOf(`**str**`), lastIndexOf(`**str**`)`	first/last index where given string begins (or -1 if not found)
`length`	number of characters in this string
`match(`**regexp**`)`	whether this string matches the given regular expression
`replace(`**str1**`, `**str2**`)`	replace all occurrences in this string of *str1* with *str2*
`slice(`**i**`, `**j**`), substring(`**i**`, `**j**`)`	characters in this string from index *i* (inclusive) to *j* (exclusive)
`split(`**delimiter**`)`	break apart string into an array of smaller strings
`toLowerCase(), toUpperCase()`	a new string with all lowercase or uppercase letters

Arrays

```
var name = [expression, expression, ..., expression];
name[index] = expression;
```

Array Method	Description
`concat(`**expr**`...)`	returns new array with appended elements/arrays
`filter(`**function**`)   *`	returns new array of elements for which function returns true
`indexOf(`**expr**`), lastIndexOf(`**expr**`)`	index of first/last occurrence of a value; -1 if not found
`join(`**separator**`)`	glue elements together into a string
`map(`**function**`)      *`	returns new array from calling the function on each element
`pop()`	remove and return last element
`push(`**expr**`...)`	append value(s) to end of array
`reduce(`**function** `[, `**initialValue**`]),` `reduceRight(`**function** `[, `**initVal**`])`	left/right reduce of array using function to combine pairs
`reverse()`	reverse order of elements, in place
`shift()`	remove and return first element
`slice(`**start**`, `**end**`)`	sub-array from start (inclusive) to end (exclusive)
`sort()` `sort(`**compareFunction**`)`	sort array in place, using optional compare function that takes 2 values and returns < 0, 0, or > 0 (a la `compareTo`)
`splice(`**index**`, `**count**`, `**expr**`...)`	remove count elements from index, then add each of the given values there
`toString()`	string representation of array, e.g. "`10,20,30,40`"
`unshift(`**expr**`...)`	insert value(s) at front of array

Objects

```
var name = {name: expression, ..., name: expression};
name.property
name["property"]
delete name.property;
```

CSE 341 Section Handout #9
JavaScript Problems

1. What is the result of each expression? Write a value of the proper type (strings in quotes, etc.).

```
a.  3 / 2 + 1 / 2                  h.  "6" < 10
b.  1.0 * 42                       i.  3 === "3"
c.  3 + "2" + 2                    j.  0 || 4 || 5
d.  3 * "2" + 3 / "2"              k.  "a" && null && "b"
e.  parseInt("3 stooges")          l.  typeof("hi")
f.  parseInt("Fantastic 4")        m.  typeof([1, 2, 3])
g.  1 + 1 == "2"                   n.  typeof(undefined)
```

2. Define a function named `countChar` that takes a string and a character as parameters and returns the number of occurrences of that character in the string, case-insensitively. For example, the call of `countChar("MiSsisSiPpI", "s")` should return `4`.

3. Define a function named `stutter` that takes an array parameter and returns the array obtained by replacing every value in the list with two of that value. For example, the call of `stutter([1, 2, 3])` should return `[1, 1, 2, 2, 3, 3]`.

4. Define a function named `sortByLength` that takes an array of strings and rearranges the strings in the array to be sorted in ascending order by length. For example, the call of `sortByLength(["bye", "hi", "goodbye", "hello", "x"])` should change the array to store `["x", "hi", "bye", "hello", "goodbye"]`. Do not define any other global symbols, but you can define local ones.

5. Define a function named `cycle` that takes array and an integer *n* and rearranges its elements by moving the first *n* values to the end of the array. For example, `cycle(4, [1, 2, 3, 4, 5, 6])` should change the array to store `[5, 6, 1, 2, 3, 4]`. If *n* is negative or 0, do not modify the array. (Bonus points if you can do it without using a loop!)

6. Define a function named `capitalizeAll` that takes array of strings and returns a new array containing all of the strings from the original array in upper case. Solve this problem using **higher-order functions** such as map/filter/reduce. For example, `capitalizeAll(["how", "are", "you?"])` should return `["HOW", "ARE", "YOU?"]`.

7. Define a **variadic** (var-args) function named `longest` that takes any number of strings as parameters and returns the longest string that had the largest length. For example, `longest("bye", "hi", "goodbye", "hello", "x")` should return `"goodbye"`. Break ties by choosing the string that occurred earlier. The call of `longest("Jen", "Tom")` should return `"Jen"`. If no arguments are passed, return the empty string, `""`. Notice that the arguments were not passed as an array. Also note that the var-args `arguments` "array" is (unfortunately) a duck-typed object and is therefore missing the array methods shown on this document's cheat sheet.

8. Define a function named `transfer` that takes two bank account objects and a real number as parameters and transfers an amount of money from the first account to the second. A bank account in this case is an object that has a numeric `balance` property. Transferring the money consists of withdrawing it from the first account's balance and depositing it in the second account's balance. Only complete the transaction if both account objects have a numeric `balance` property and the first account has at least the given amount of money to be withdrawn. If the transaction was successful, return `true`, otherwise `false`.

CSE 341 Section Handout #9
Solutions

1.

	Expression	Result
a.	`3 / 2 + 1 / 2`	`2`
b.	`1.0 * 42`	`42`
c.	`3 + "2" + 2`	`"322"`
d.	`3 * "2" + 3 / "2"`	`7.5`
e.	`parseInt("3 stooges")`	`3`
f.	`parseInt("Fantastic 4")`	`NaN`
g.	`1 + 1 == "2"`	`true`
h.	`"6" < 10`	`true`
i.	`3 === "3"`	`false`
j.	`0 || 4 || 5`	`4`
k.	`"a" && null && "b"`	`null`
l.	`typeof("hi")`	`"string"`
m.	`typeof([1, 2, 3])`	`"object"`
n.	`typeof(undefined)`	`"undefined"`

2.

```
function countChar(s, character) {
    s = s.toLowerCase();
    character = character.toLowerCase();
    var count = 0;
    for (var i = 0; i < s.length; i++) {
        if (s[i] == character) {
            count++;
        }
    }
    return count;
}
```

3.

```
function stutter(a) {
    var result = [];
    for (var i = 0; i < a.length; i++) {
        result.push(a[i]);
        result.push(a[i]);
    }
    return result;
}
```

4.

```
function sortByLength(strings) {
    function compareLength(s1, s2) {
        return s1.length - s2.length;
    };
    strings.sort(compareLength);
}
```

CSE 341 Section Handout #9
Solutions (continued)

5.

```
function cycle(a, n) {
    for (var i = 0; i < n; i++) {
        a.unshift(a.pop());
    }
}

function cycle(a, n) {    // UB3R-1337 solution
    a.concat(a.slice(n), a.slice(0, n));
}
```

6.

```
function capitalizeAll(strings) {
    return strings.map(function(s) { return s.toUpperCase(); });
}
```

7.

```
function longest() {
    var result = "";
    for (var i = 0; i < arguments.length; i++) {
        if (arguments[i].length > result.length) {
            result = arguments[i];
        }
    }
    return result;
}
```

8.

```
function transfer(acct1, acct2, money) {
    if (typeof(acct1.balance) === "number" &&
            typeof(acct2.balance) === "number" &&
            acct1.balance >= money) {
        acct1.balance -= money;
        acct2.balance += money;
        return true;
    } else {
        return false;
    }
}
```