# Web Applications 01 Cheat Sheet

by alex09998 via cheatography.com/137559/cs/42627/

## General info

We use ES6 (EcmaScript standard version 6)

Runs natively on browsers AND on a computer (using Node.js). Execution environments: Server & CLI Node.js, Browser, learning aids (Python Tutor).

JS engines (interpreters): V8, SpiderMonkey, JavaScriptCore.

**Compatibility:**

- Backwards: https://babeljs.io/, Polyfilling
- Strict mode disables dangerous old semantics

First line of file is

```
"use strict " ;
```

- Can't define properties/parameters with same name;

## Control Structures

If conditions: `if - else if - else`, `switch (expr)`

Loops: `for (initi al_ expr; cond; increm ent;)`, `do - while`, `while`

Special **for** statements:

`for (var in object) {}`: iterates over all the enumerable **properties** of an object. Do **not** use to traverse an array.

`for (var of iterable) {}`: iterates the variable over all values of an **iterable object** (array, map, etc.) and returns **the values**.

**Exception handling**:

`try-ca tch -fi nally.`

Ready to use throwables.

The condition of the ifs causes an implicit conversion of whatever is written to a boolean.

The expression in the switch may also be a string.

In loops, we may use `break;` or `continue;`

## Expressions

Declare + initialize:

```
let variable = expression ;
```

Reassign:

```
variable = expression ;
```

**Comparison**:

convert and compare: `a == b`

same type **and** value: `a === b`

**Conversions**:

any to boolean:

```
truthy -falsy rule, Boolea n(a), !!a
```

String to Number:

```
Number(s), parseI nt(s), parseF loat(s)
```

Number to String:

```
n.toSt ring(), String(n), n+""
```

String **concatenation**:

```
string1 + string2
```

**Default value** assignment (*if a then a else b*):

```
a || b
```

## Strings

Immutable sequence of unicode characters. All **operations** always return **new strings**.

Length = # of characters (not bytes).

Empty string has length 0 and is a **falsy**.

**Operations**:

- ➔ indexing `s[3]`
- ➔ concatenation `s1 + s2`
- ➔ # of characters `s.length`

**Template literals ("dynamic string concatenation")**

```
let name = " Bil l";
let greeting = `Hello ${ name }.`;
```

Some Unicode characters are represented by **two** code units, so some string methods above FFFF might misbehave.

## Language Structure

**One file = one JS program**
((loaded independently but communicate w/ global state and modules)

File is entirely **parsed** and **then** executed top to bottom.

Written in **Unicode**, case sensitive.

## Types and Variables

**Values** have **types**:
"type" is a property of a value. {{nl}}
**Variables** DON'T have types:
variables can contain any type, and ≠ types in ≠ moments.

Boolean type:
'true' or 'false' literal values
**Conversion:**
**Truthies:** `0, -0, NaN, undefined, null, ' '`
**Falsies:** `3, 'false', [], {}, ...`

Numbers:
→ No distinction integers and reals
→ **Automatic conversion** according to operation

**Nullish values**:
→ `undefined`: variable declared but not initialized. Returned by void functions.
→ `null`: empty value

**Variables**:
→ They're pure references: refer to a **value**
→ Declare: **let**, **const**, var.
→ `Let`: **yes** reassign, no redeclare, block scope, no hoisting
→ `Const`: **no** reassign, no redeclare, block scope, no hoisting
→ `Var`: yes reassign and redeclare, function/global scope, hoisting. {{ln}} {{ln}} Block scope: variable exists only in defined and inner scopes. {{ln}} **Hoisting**: declaration of var inside code is moved to top of scope.

## Arrays

Elements do **not** need to be of the same **type**.

Have property `length` (automatic).

**Create** arrays using parameters:
`let v = Array.o f(1, 2, 3) ;`

**Add elements**:
`let v = [] ; v[0] = " a" ; v[1] = 8 ;`
`.push()` adds to end of array
`.unshift()` adds to beginning of array.
`.length` adjusts **automatically**.

**Removal**: `.unshift()` and `.pop()`.

**Copy of the reference**: `let v = []; let alias = v;` we establish `alias` as an alias of `v`, so if we modify `alias` we're actually modifying `v`.

**Shallow copy of arrays**:
`let copy = Array.f ro m(v);`

**Destructuring assignment** :
Value of the right are extracted and stored in the variables on the left. `[x,y] = [y,x];` **easy swap**.

**Spread operator** :
→ "all the rest": `let [x, ...y] = [1,2,3 ,4];` · we obtain y == [2,3,4]
→ "everything inside vector x".
Can be used to **copy arrays by value**: `const b = Array.o f(.. .a), const b = [...a]`

Automatic expansion of array: `let v = []; v[3] = " a".`

Arrays are not values, they're **references**.