Cheatography

JS Cheat Sheet

by milad69_1 via cheatography.com/52248/cs/15235/

Basics

On page script:

```
<script type="text/javascript"> ... </script>
```

Include External script:

```
<script src="filename.js"></script>
```

Deter

Use deter so that the JavaScript doesn't execute until after the DOM is loaded

```
<script src="script.js" defer></script>
```

Comments:

```
/* Multi line
   comment */
// One line
```

Outputs:

```
console.log(a); // write to the browser console
document.write(a); // write to the HTML
alert(a); // output in an alert box
confirm("Really?"); // yes/no dialog, returns true/false depending on user click
prompt("Your age?","0"); // input dialog. Second argument is the initial value
```

Data Types

```
var age = 18; // number
var name = "Jane"; // string
var name = {first:"Jane", last:"Doe"}; // object
var truth = false; // boolean
var sheets = ["HTML","CSS","JS"]; // array
var a; typeof a; // undefined
var a = null; // value null
Symbol type???
```

Objects:

```
var student = { // object name
    firstName:"Jane", // list of properties and values
    lastName:"Doe",
    age:18,
    height:170,
    fullName : function() { // object function
       return this.firstName + " " + this.lastName;
    }
};
student.age = 19; // setting value (add a property)
student[age]++; // incrementing
delete student.age; // delete a property
name = student.fullName(); // call object function
iterating:
for (key in object){
object[key];
}
```

Data Types evaluation

```
'' == '0' // false
'' == 0 // true
0 == '0' // true
NaN == NaN // false
[''] == '' // true
false == undefined // false
false == null // false
null == undefined // true
'' === '0' // false
'' === 0 // false
0 === '0' // false
NaN == NaN // still weirdly false
[''] === '' // false
false === undefined // false
false === null // false
null === undefined // false
```

Dom Elements

Access an HTML element

```
document.querySelector('css selector');
document.querySelectorAll('css selector');
document.getElementById("elementID").innerHTML =
"Hello World!";
example:
let element = document.querySelector('#button');
let allElements = document.querySelectorAll('css
selector');
let elems = document.querySelectorAll('.quote,
.comment'); // that have a
"quote" class AND all elements that have a "comment"
class.
element.style.top = '20px';
element.style.
```

Adding event listeners

Use dom function:

```
addEventListener(event name, function name);
```

Access attributes

you can't access the class attribute via object.class

```
const image = document.querySelector('img');
image.src ='somthing.jpg';
```

but you can't access the class attribute via object.class

Access class attribute

classList.add and classList.remove:

```
const image = document.querySelector('img');
image.classList.add('active');
image.classList.remove('hidden');
```

innerHTML: The raw HTML between the starting and ending tags of an element, as a string

textContent: The text content of a node and its descendants. (This property is inherited from Node)

Create and remove elements

```
element = document.createElement(tag string)
element = document.createElement('h1')
containerElement.appendChild(element);
```

Dom Elements (cont)

```
element.remove();
```

Style

Every element has a style attribute which has higher precedence than any css file.

```
element.style.font = 'something'
```

To undo a style set via the style attribute, you can set it to the empty string, the element will be styled according to any rules in the CSS file

```
element.style.font = ''
```

Conditionals

if else

```
if ((age >= 14) && (age < 19)) { // logical condition
    status = "Eligible."; // executed if condition is
true
} else { // else block is optional
    status = "Not eligible."; // executed if condition
is false
}
```

Switch statement:

```
switch (new Date().getDay()) { // input is current day
    case 6: // if (day == 6)
        text = "Saturday";
        break;
    case 0: // if (day == 0)
    case 1: // if (day == 1)
        text = "Sunday";
        break;
    default: // else...
        text = "Whatever";
}
```

? statements:

```
a = (condition) ? (cond_true_result) :
(cond_false_result);
```

Numbers and Math

```
var pi = 3.141;
pi.toFixed(0); // returns 3
pi.toFixed(2); // returns 3.14 - for working with
money
pi.toPrecision(2) // returns 3.1
pi.valueOf(); // returns number
Number(true); // converts to number
Number(new Date()) // number of milliseconds since
1970
parseInt("3 months"); // returns the first number: 3
parseFloat("3.5 days"); // returns 3.5
Number.MAX_VALUE // largest possible JS number
Number.MIN_VALUE // smallest possible JS number
Number.NEGATIVE_INFINITY// -Infinity
Number.POSITIVE_INFINITY// Infinity
```

Math

```
var pi = Math.PI; // 3.141592653589793
Math.round(4.4); // = 4 - rounded
Math.round(4.5); // = 5
Math.pow(2,8); // = 256 - 2 to the power of 8
Math.sqrt(49); // = 7 - square root
Math.abs(-3.14); // = 3.14 - absolute, positive value
Math.ceil(3.14); // = 4 - rounded up
Math.floor(3.99); // = 3 - rounded down
Math.sin(0); // = 0 - sine
Math.cos(Math.PI); // OTHERS: tan,atan,asin,acos,
Math.min(0, 3, -2, 2); // = -2 - the lowest value
Math.max(0, 3, -2, 2); // = 3 - the highest value
Math.log(1); // = 0 natural logarithm
Math.exp(1); // = 2.7182pow(E,x)
Math.random(); // random number between 0 and 1
Math.floor(Math.random() * 5) + 1; // random integer,
from 1 to 5
```

Constants:

Numbers and Math (cont)

```
E, PI, SQRT2, SQRT1_2, LN2, LN10, LOG2E, Log10E
```

JSON

```
var str = '{"names":[' + // crate JSON object
'{"first":"Hakuna","lastN":"Matata" },' +
'{"first":"Jane","lastN":"Doe" },' +
'{"first":"Air","last":"Jordan" }]}';
obj = JSON.parse(str); // parse
document.write(obj.names[1].first); // access
```

Send:

```
var myObj = { "name":"Jane", "age":18,
"city":"Chicago" }; // create object
var myJSON = JSON.stringify(myObj); // stringify
window.location = "demo.php?x=" + myJSON; // send to
php
```

Store:

```
myObj = { "name":"Jane", "age":18, "city":"Chicago" };
myJSON = JSON.stringify(myObj); // storing data
localStorage.setItem("testJSON", myJSON);
////
text = localStorage.getItem("testJSON"); // retrieving
data
obj = JSON.parse(text);
document.write(obj.name);
```

Errors

```
try { // block of code to try
    undefinedFunction();
}
catch(err) { // block to handle errors
    console.log(err.message);
}
```

Throw Error

```
throw "My error message"; // throw a text
```

Input Validation

Errors (cont)

```
var x = document.getElementById("mynum").value; // get input value
try {
    if(x == "") throw "empty"; // error cases
    if(isNaN(x)) throw "not a number";
    x = Number(x);
    if(x > 10) throw "too high";
}
catch(err) { // if there's an error
    document.write("Input is " + err); // output error
    console.error(err); // write the error in console
}
finally {
    document.write("</br />Done"); // executed regardless of the try / catch result
}
```

Error name values

RangeError A number is "out of range"
ReferenceError An illegal reference has occurred
SyntaxError A syntax error has occurred
TypeError A type error has occurred
URIError An encodeURI() error has occurred

Cite

Got from http://htmlcheatsheet.com/js/ with customizations

Event types

Mouse

onclick, oncontextmenu, ondblclick, onmousedown, onmouseenter, onmouseleave, onmousemove(only desktop), onmouseover, onmouseout, onmouseup
mouseEvent.clientX, mouseEvent.clientY: X|Y axis position relative to the left edge of the browser viewpoint

Touch

ontouchcancel, ontouchend, ontouchmove(only mobile), ontouchstart

Event types (cont)

PointerEvent

(not for all browser can be used with polyfill)
available with the following script:

```
<script src="https://code.jquery.com/pep/0.4.1/pep.js"></script>
```

And we'll add need to add touch-action="none" to the area where we want PointerEvents to be recognized (telling the browser that we do not want the default touch behavior for children of this element).
Works the same both on mobile and desktop
pointerdown, pointerup, pointermove(works both on mobile and desktop), pointercancel
To listen to pointer events that occur when the pointer goes offscreen, call setPointerCapture on the target you want to keep tracking:

```
event.target.setPointerCapture(event.pointerId);
```

Keyword

onkeydown, onkeypress, onkeyup
keywordEvent.key: which key was pressed: Escape, ArrowRight, ArrowLeft, ...

Frame

onabort, onbeforeunload, onerror, onhashchange, onload, onpageshow, onpagehide, onresize, onscroll, onunload

Form

onblur, onchange, onfocus, onfocusin, onfocusout, oninput, oninvalid, onreset, onsearch, onselect, onsubmit

Drag

ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop

Clipboard

oncopy, oncut, onpaste

Media

onabort, oncanplay, oncanplaythrough, ondurationchange, onended, onerror, onloadeddata, onloadedmetadata, onloadstart, onpause, onplay, onplaying, onprogress, onratechange, onseeked, onseeking, onstalled, onsuspend, ontimeupdate, onvolumechange, onwaiting

Animation

animationend, animationiteration, animationstart

Event types (cont)

Miscellaneous

transitionend, onmessage, onmousewheel, ononline, onoffline, onpopstate, onshow, onstorage, ontoggle, onwheel

Working with Events

Adding event listeners

Use dom function:

addEventListener(event name, function name);

Access caller element inside callback

event.target; //the element that was clicked / "dispatched the event" (might be a child of the target)

event.currentTarget; // the element that the original event handler was attached to)

also 'this' refer to event in callback

Event propagation

If a child and a parent element exist that have different callbacks to events, and inner element is clicked, both events are called. This is called event bubbling.

```
<div id="outer">
  Click me!
  <div id="inner">
    No, Click me!
  </div>
</div>
```

```
innerElem.addEventListener('click', innerCallback);
outerElem.addEventListener('click', oute/rCallback);
```

if inner element is called, outer and inner callbacks will be fired.

event.stopPropagation() method inside callback can prevent further propagation of event to outer callbacks.

the propagation can go in opposite direction (called event capturing) by setting additional parameter for addEventListener.

```
event.addEventListener('click', onClick, { capture: true} );
```

prevent default

There's a default behavior for some events that can be prevented using preventDefault method.

Working with Events (cont)

```
function onCallback(event){
event.preventDefault();
}
```

Random number

Use Math.random: returns a random floating point number between [0, 1)

To get a random number in [0, maxNumber):

```
Math.floor(Math.random() * Max)
```

Data attributes

You can assign special data-* attributes to HTML elements to give associate additional data with the element

```
<tag data-your-name='your value"> </tag>
```

to access in javascript:

```
tagElement.dataset.YourName
```

Dash-separated words turn to camel case and are returned as string

Global functions

```
eval(); // executes a string as if it was script code
String(23); // return string from number
(23).toString(); // return string from number
Number("23"); // return number from string
decodeURI(enc); // decode URI. Result: "my page.asp"
encodeURI(uri); // encode URI. Result: "my%page.asp"
decodeURIComponent(enc); // decode a URI component
encodeURIComponent(uri); // encode a URI component
isFinite(); // is variable a finite, legal number
isNaN(); // is variable an illegal number
parseFloat(); // returns floating point number of string
parseInt(); // parses a string and returns an integer
```

Dates

```
Thu Mar 22 2018 11:40:54 GMT+0430 (+0430)
var d = new Date();
1521702654616 miliseconds passed since 1970
Number(d)
Date("2017-06-23"); // date declaration
Date("2017"); // is set to Jan 01
Date("2017-06-23T12:00:00-09:45"); // date - time
YYYY-MM-DDTHH:MM:SSZ
Date("June 23 2017"); // long date format
Date("Jun 23 2017 07:45:00 GMT+0100 (Tokyo Time)"); //
time zone
Get Dates:
Thu Mar 22 2018 11:40:54 GMT+0430 (+0430)
var d = new Date();
1521702654616 miliseconds passed since 1970
Number(d)
Date("2017-06-23"); // date declaration
Date("2017"); // is set to Jan 01
Date("2017-06-23T12:00:00-09:45"); // date - time
YYYY-MM-DDTHH:MM:SSZ
Date("June 23 2017"); // long date format
Date("Jun 23 2017 07:45:00 GMT+0100 (Tokyo Time)"); //
time zone
Setting part of a date:
var d = new Date();
d.setDate(d.getDate() + 7); // adds a week to a date
setDate(); // day as a number (1-31)
setFullYear(); // year (optionally month and day)
setHours(); // hour (0-23)
setMilliseconds(); // milliseconds (0-999)
setMinutes(); // minutes (0-59)
setMonth(); // month (0-11)
setSeconds(); // seconds (0-59)
setTime(); // milliseconds since 1970)
```

Regular Expressions

Promises

```
An object used to manage asynchronous results.
has a then() method that lets you attach functions to
execute onSuccess and onError callbacks.
Allows you to build chains of asynchronous results.
```

Local Storage

Variables

```
var a; // Function scope variable
let al; // block scope variable
const ca; // block scope constant, cannot be changed.
However, it doesn't provide true const correctness, so
you can still modify the underlying object so:
const list = [1, 2, 3];
list.push(4); // OK
var b = "init"; // string
var c = "Hi" + " " + "Joe"; // = "Hi Joe"
var d = 1 + 2 + "3"; // = "33"
var e = [2,3,5,8]; // array
var f = false; // boolean
var g = /()/; // RegEx
var h = function(){}; // function object
const PI = 3.14; // constant
var a = 1, b = 2, c = a + b; // one line
let z = 'zzz'; // block scope local variable
Strict mode
"use strict"; // Use strict mode to write secure code
x = 1; // Throws an error because variable is not
declared
```

Operators

Values

```
false, true // boolean
18, 3.14, 0b10011, 0xF6, NaN // number
"flower", 'John' // string
undefined, null , Infinity // special
```

Operators

```
a = b + c - d; // addition, substraction
a = b * (c / d); // multiplication, division
x = 100 % 48; // modulo. 100 / 48 remainder = 4
a++; b--; // postfix increment and decrement
```

Bitwise operators

```
& AND 5 & 1 (0101 & 0001) 1 (1)
| OR 5 | 1 (0101 | 0001) 5 (101)
~ NOT ~ 5 (~0101) 10 (1010)
XOR 5 1 (0101 ^ 0001) 4 (100)
<< left shift 5 << 1 (0101 << 1) 10 (1010)
>> right shift 5 >> 1 (0101 >> 1) 2 (10)
>>> zero fill right shift 5 >>> 1 (0101 >>> 1) 2 (10)
```

Asthmatics:

```
a * (b + c) // grouping
person.age // member
person[age] // member
!(a == b) // logical not
a != b // not equal
typeof a // type (number, object, function...)
x << 2 x >> 3 // minary shifting
a = b // assignment
a == b // equals
a != b // unequal
a === b // strict equal
a !== b // strict unequal
```

Operators (cont)

```
a < b a > b // less and greater than
a <= b a >= b // less or equal, greater or eq
a += b // a = a + b (works with - * %...)
a && b // logical and
a || b // logical or
```

Strings

```
var abc = "abcdefghijklmnopqrstuvwxyz";
var esc = 'I don\'t \n know'; // \n new line
var len = abc.length; // string length
abc.indexOf("lmno"); // find substring, -1 if doesn't
contain
abc.lastIndexOf("lmno"); // last occurance
abc.slice(3, 6); // cuts out "def", negative values
count from behind
abc.replace("abc","123"); // find and replace, takes
regular expressions
abc.toUpperCase(); // convert to upper case
abc.toLowerCase(); // convert to lower case
abc.concat(" ", str2); // abc + " " + str2
abc.charAt(2); // character at index: "c"
abc[2]; // unsafe, abc[2] = "C" doesn't work
abc.charCodeAt(2); // character code at index: "c" ->
99
abc.split(","); // splitting a string on commas gives
an array
abc.split(""); // splitting on characters
128.toString(16); // number to hex(16), octal (8) or
binary (2)
```

Functions

Declaration

```
functions in JavaScript are "first-class" because they
are treated like any other variable/object.
function hello(input) {
// body
}
hello = (input) => {
// body
```

Functions (cont)

```
}
// you can omit {} if there is only one statement:
// you can omit () if there is only one argument
hello = input => input + 1;
var hello = function hello(input) { // body }
```

this scope inside functions

functions:
this is will be dynamically assigned to a different value depending on how the function is called (unless explicitly bound with bind)
- this in a constructor: refers to the new object being
created
- this outside a constructor: refers to a different value
depending on how the function is called
  - In response to a DOM event, this is the element that the event handler was tied to
  - When called in a method, this is the object that the method is called from
- bind: sets the value of this for a function so it does
not change depending on the context
Arrow functions link:
Arrow functions capture the this value of the enclosing context.
Arrow function syntax uses “lexical scoping”. Lexical scoping is fancy way of saying it uses “this” from the surrounding code… the code that contains the code in question.
Arrow functions do not have their own this.
arrow functions cannot be used as generators (yeild does not work unless for further nested function)

```
this.test = "attached to the module";
var foo = {
  test: "attached to an object"
};
foo.method = function(name, cb){
  this[name] = cb;
};
foo.method("bar", () => {
  // not what you expected, maybe?
  console.log(this.test);
});
```

foo.bar(); // => "attached to an object" but it was expected to be madule!
As another example:

```
var obj = {
  i: 10,
  b: () => console.log(this.i, this),
  c: function() {
    console.log(this.i, this);
  }
}
obj.b(); // prints undefined, Window {...} (or the global object)
obj.c(); // prints 10, Object {...}
```

this inside object as function:

```
obj = { 'a': 12, f: function(){
console.log(this.a)}}; obj.f() // => 12
```

this inside object as arrow function:

```
obj = { 'a': 12, f: () => { console.log(this.a)}};
obj.f(); // => undefined.
```

or:

```
a = 1; obj = { 'a': 12, f: () => {
console.log(this.a)}}; obj.f(); // => 1
```

functional programming:

- First-class functions (functions as objects)
- Closures
- Anonymous functions / lambdas / function literals
- Currying

Hoisted declaration

Some function declaration are hoisted. definition gets moved to the top of the scope in which it's defined. So this works:

Functions (cont)

```
hello()
function hello() { // ,,, }
```

These kind of declaration are not hoisted:

closures

A function declared within a function is also known as a closure. Inner functions are called closures

```
function outerFunc() {
 function innerFunc() {
 // closures
 }
}
```

Scope:

Functions declared with function (or var) have function scope

Functions declared with const or let have block scope

Access to outer function variables

closures have access to variables declared in outer function and those variables does not go away! (remains and can be even changed!!). It's like a function pointer but local variables of outer function remained and are not destroyed.

```
f = function outerFunction(){
 let x = 1;
 return function(){
 x = x + 1;
 console.log(x);
 }
}
run = f();
run(); // 2
run(); // 3
run(); // 4
```

curryng

Functions (cont)

The idea of constructing a new function that is "partially instantiated" with arguments is called currying.

Constructing a new function that references part of the outer function's parameters is called currying. for example when one function needs additional argument and we can't add another argument to it (it's not expected to have another one!), we can use currying. we can have a function that accepts that additional argument and have another function (a closure) which has expected arguments:

we want a function with 2 arguments to have another argument:

```
somthing.add(twoArgumentFunc)
function makeTwoArgument(anotherArg){
  return function(){
    // use anotherArg in here
    // It's like having 3 arguments!
  }
}
=> somthing.add(makeTwoARgument(anotherArg));
```

Anonymous function

When we define a function without an identifier, we call it an anonymous function

```
return function() {
 //this is anonymous function
}
```

Arrays

```
var dogs = ["Bulldog", "Beagle", "Labrador"];
var dogs = new Array("Bulldog", "Beagle",
"Labrador"); // declaration
alert(dogs[1]); // access value at index, first item
being [0]
dogs[0] = "Bull Terier"; // change the first item
for (var i = 0; i < dogs.length; i++) { // parsing
with array.length
    console.log(dogs[i]);
}
```

Methods:

```
dogs.toString(); // convert to string: results
"Bulldog,Beagle,Labrador"
```

Arrays (cont)

```
dogs.join(" "); // join: "Bulldog Beagle * Labrador"
dogs.pop(); // remove last element
dogs.push("Chihuahua"); // add new element to the end
dogs[dogs.length] = "Chihuahua"; // the same as push
dogs.shift(); // remove first element
dogs.unshift("Chihuahua"); // add new element to the beginning
delete dogs[0]; // change element to undefined (not recommended)
dogs.splice(2, 0, "Pug", "Boxer"); // add elements (where, how many to remove, element list)
var animals = dogs.concat(cats,birds); // join two arrays (dogs followed by cats and birds)
dogs.slice(1,4); // elements from [1] to [4-1]
dogs.sort(); // sort string alphabetically
dogs.reverse(); // sort string in descending order
x.sort(function(a, b){return a - b}); // numeric sort
x.sort(function(a, b){return b - a}); // numeric descending sort
highest = x[0]; // first item in sorted array is the lowest (or highest) value
x.sort(function(a, b){return 0.5 - Math.random()}); // random order sort
splice // Add/remove element at index
list.splice(startIndex, deleteCount, item1, item2, ...)
list.splice(3, 1) //Remove one element at index 3:
list.splice(2, 0, element); //Add element at index 2:
list.forEach(function): Executes the provided function once for each array element.
list.filter(function): Creates a new array with all elements that pass the test implemented by the provided function.
list.every(function) : Tests whether all elements in the array pass the test implemented by the provided function
others:
 concat, copyWithin, every, fill, filter, find, findIndex, forEach, indexOf, isArray, join, lastIndexOf, map, pop, push, reduce, reduceRight, reverse, shift, slice, some, sort, splice, toString, unshift, valueOf
```

Loops

For Loops:

```
for (var i = 0; i < 10; i++) {
    document.write(i + ": " + i*3 + "<br />");
}
var sum = 0;
for (var i = 0; i < a.length; i++) {
    sum + = a[i];
} // parsing an array
```

For each

```
for ( var i in something){
  console.log(something[i]);
}
```

While Loops:

```
var i = 1; // initialize
while (i < 100) { // enters the cycle if statement is true
    i *= 2; // increment to avoid infinite loop
    document.write(i + ", "); // output
}
```

Do While Loops:

```
var i = 1; // initialize
do { // enters cycle at least once
    i *= 2; // increment to avoid infinite loop
    document.write(i + ", "); // output
} while (i < 100) // repeats cycle if statement is true at the end
```

Break:

```
for (var i = 0; i < 10; i++) {
    if (i == 5) { break; } // stops and exits the cycle
    document.write(i + ", "); // last output number is 4
}
```

Continue:

```
for (var i = 0; i < 10; i++) {
```

Loops (cont)

```
    if (i == 5) { continue; } // skips the rest of the
cycle
    document.write(i + ", "); // skips 5
}
```

Classes

Definition

```
class ClassName{
  constructor(){ //it's optional
    // define public fields by setting this.fieldName
    // in constructor or any method
    this.fieldName = value;

  }

  methodName(){ //don't need to use function keyword
    //all methods are public - no private method yet
    // convention: _ before the name means private
  }
  methodTwo(){
  //always refer to other methods or fields in class
with this
    this.methodName();

  }
}
```

Element as a class

Use class that gets Container element as input for constructor

Definition

```
class ClassName{
  constructor(containerElement){
    this.containerElem = containerElement;

    //Create element:
    this.image = document.createElement('img');
    this.image.src = 'image.jpg';
    this.image.addEventListener('click',
this._clickElement)

  }
```

Classes (cont)

```
  _clickElement(event){
    // do some thing
  }
}
```

Binding this

This in here refers to the element that the event handler was attached to!

to make this always refer to the instance object for a method in the class we use bind

for above example:

first bind this in the constructor:

```
 constructor(containerElement){
 // always bind event listeners in the constructor
  this._clickElement = this._clickElement.bind(this);
 }
```

then use this as expected:

```
clickElement(event){
// now this refer to the class object not event
element!
    this.image.src = 'image2.jpg';
}
```

another solution was to access the element with event.currentTarget

Instantiation

```
const x = new SomeClass();
const y = new SomeClass();
y.someMethod();
```

Custom events

You can listen to and dispatch (fire) Custom Events.

```
const event = new CustomEvent(eventNameString,
optionalParameterObject);
element.addEventListener(eventNameString);
element.dispatchEvent(eventNameString);
```

CustomEvent can only be listened to /dispatched on HTML elements, and not to arbitrary class instances.

Communicating between element

Custom events (cont)

```
You can add custom event on the document and dispatch
that event inside different element.
document.addEventListerner( ...
document.dispatchEvent(...
to add additional information about the event you can
pass parameter to the custom event:
onClick(event) {
 const eventInfo = {
 buttonName = this.text;
 }
 document.dispatchEvent( new CustomEvent('bclicked', {
detail: eventinfo});
}
onCustomEven(event) {
event.detail.buttonName; // can access info
}
```

Template literals

```
Template literals allow you to embed expressions in
JavaScript strings
const port = 80;
console.log(Server is listening on port: ${port} );
```

async/await

Fetch

```
The API to use to load external resources (text, JSON,
etc) in the browser (like XHR but easier to use)
- It has only one function: fetch
- takes string path to the resource as argument
- return a promise
Fetch api:
function onSuccess(response) {
...
}
function onFail(response) {
```

Fetch (cont)

```
...
}
fetch('images.txt').then(onSuccess, onFail);
or
const promise = fetch('images.txt');
promise.then(onSuccess, onFail);
```