thegitfather / **vanilla-js-cheatsheet.md**

Last active 3 days ago • Report gist

Vanilla JavaScript Quick Reference / Cheatsheet

vanilla-js-cheatsheet.md

Vanilla JavaScript Quick Reference / Cheatsheet

Just migrated it from Codepen.io to markdown. Credit goes to David Conner.

Working with DOM	Working with JS	Working With Functions
Accessing Dom Elements	Add/Remove Array Item	Add Default Arguments to Function
Grab Children/Parent Node(s)	Add/Remove Object Properties	Throttle/Debounce Functions
Create DOM Elements	Conditionals	
Add Elements to the DOM	Loops	
Add/Remove/Toggle/Check Classes	Events	
	Timers	
	Type Checking	

Accessing Dom Elements

```
// Returns a reference to the element by its ID.
document.getElementById('someid');

// Returns an array-like object of all child elements which have all of the given class names.
document.getElementsByClassName('someclass');

// Returns an HTMLCollection of elements with the given tag name.
document.getElementsByTagName('LI');

// Returns the first element within the document that matches the specified group of selectors.
document.querySelector('.someclass');

// Returns a list of the elements within the document (using depth-first pre-order traversal of the document'
// that match the specified group of selectors.
document.querySelectorAll('div.note, div.alert');
```

Grab Children/Parent Node(s)

```
// Get child nodes
var stored = document.getElementById('someid');
var children = stored.childNodes;

// Get parent node
var parental = children.parentNode;
```

Create New DOM Elements

```
// create new elments
var newHeading = document.createElement('h1');
var newParagraph = document.createElement('p');

// create text nodes for new elements
var h1Text= document.createTextNode('This is a nice header text!');
var pText= document.createTextNode('This is a nice paragraph text!');
```

```
// attach new text nodes to new elements
newHeading.appendChild(h1Text);
newParagraph.appendChild(pText);

// elements are now created and ready to be added to the DOM.
```

Add Elements to the DOM

```
// grab element on page you want to add stuff to
var firstHeading = document.getElementById('firstHeading');

// add both new elements to the page as children to the element we stored in firstHeading.
firstHeading.appendChild(newHeading);
firstHeading.appendChild(newParagraph);

// can also insert before like so

// get parent node of firstHeading
var parent = firstHeading.parentNode;

// insert newHeading before FirstHeading
parent.insertBefore(newHeading, firstHeading);
```

Add Elements to the DOM cont.

Suppose you have the following HTML:

```
<div id='box1'>
  <p>Some example text</p>
</div>
<div id='box2'>
  <p>Some example text</p>
</div>
```

You can insert another snippet of HTML between #box1 and #box2:

```
var box2 = document.getElementById('box2');
box2.insertAdjacentHTML('beforebegin', '<div><p>This gets inserted.</p></div>');

// beforebegin - The HTML would be placed immediately before the element, as a sibling.
// afterbegin - The HTML would be placed inside the element, before its first child.
// beforeend - The HTML would be placed inside the element, after its last child.
// afterend - The HTML would be placed immediately after the element, as a sibling.
```

Add/Remove/Toggle/Check Classes

```
// grab element on page you want to use
var firstHeading = document.getElementById('firstHeading');

// will remove foo if it is a class of firstHeading
firstHeading.classList.remove('foo');

// will add the class 'anotherClass' if one does not already exist
firstHeading.classList.add('anotherclass');

// add or remove multiple classes
firstHeading.classList.add('foo', 'bar');
firstHeading.classList.remove('foo', 'bar');

// if visible class is set remove it, otherwise add it
firstHeading.classList.toggle('visible');

// will return true if it has class of 'foo' or false if it does not
firstHeading.classList.contains('foo');
```

Add/Remove Array Item

```
// create an empty array
var myArray = [];

// create array with items. Can store any type
var myOtherArray = [myArray, true, 'a random string'];

// call specific value in an array
myOtherArray[0];
// will return myArray

// change value for this item
myOtherArray[0] = false;
// will now return false


// add to end of array
myOtherArray[myOtherArray.length] = 'new stuff';
// will return the new item 'new stuff'

// or you can use push()
myOtherArray.push('new stuff');
// will return new length of array

// you can remove this last item by using pop()
myOtherArray.pop();
// will return the last item of the array and will have removed it from myOtherArray

// shift and unshift will do the same for the begging of the Array
myOtherArray.shift();
// will remove and return first item of array

myOtherArray.unshift(1,2);
// this will add 1 and 2 to beginning of array and return new length

// you can use delete keyword but turn value to undefine and not shorten length. so we use splice()
myOtherArray.splice(2, 1);
// this will remove and return  the third item only.
// first arg is where to start and second is how many things to splice. this example is 1.
```

Add/Remove Object Properties

```
// create an object
var newObject = {};

// add a property to object
newObject.newPropName = 'super slick';

// or other syntax
newObject['other new prop name'] = 'mildly slick';

// Now newObject.newPropName will return 'super slick'
newObject.newPropName;

// now to delete
delete newObject.newPropName;
```

Conditionals

```
// If Else statements
var a = 1;
var b = 2;

if (a < b) {
  console.log('the if is true!');
} else {
  console.log('the if is false!');
}


// Multi If Else statements
var a = 1;
```

```
var b = 2;
var c = 3;

if (a > b) {
  console.log('a is bigger than b');
} else if (a > c) {
  console.log('but a is bigger than c');
} else {
  console.log('a is the smallest');
}


// Ternary operators. same as if else
var a = 1;
var b = 2;

a === b ? console.log('The statement is true') : console.log('The statement is false');

// switch statements
var a = 4;
switch (a) {
  case 'Oranges':
    console.log('Orange? really?');
    break;
  case 1:
    console.log('a is equal to 1.');
    break;
  case 2:
    console.log('a is equal to 2.');
    break;
  case 3:
    console.log('a is equal to 3.');
    break;
  case 4:
    console.log('a is equal to 4.');
    break;
  default:
    console.log('I run if no one else is true.');
}
```

Loops

```
// while loop
var i = 0;
while (i < 10) {
  console.log(i);
  i += 1
}


// do while loop
var i = 0;
do {
  console.log(i);
  i += 1
} while (i < 10)


// for loop
for (var i = 0; i < 10; i++) {
   console.log(i);
}

// for in statments
var obj = {a:1, b:2, c:3};

for (var prop in obj) {
  // check if property is inherited or not
  if (obj.hasOwnProperty(prop)) {
    console.log('obj.' + prop + ' = ' + obj[prop]);
  }
}
```

Events (MDN Event reference)

```
var newElement = document.getElementsByTagName('h1');

newElement.onclick = function() {
  console.log('clicked');
};

var logEventType = function(e) {
    console.log('event type:', e.type);
};

newElement.addEventListener('focus', logEventType, false);
newElement.removeEventListener('focus', logEventType, false);

window.onload = function() {
  console.log('Im loaded');
};
```

Timers

```
function simpleMessage() {
  alert('This is just a simple alert');
}

// set time out
window.setTimeout(simpleMessage, 5000);

// if you wanted to clear the timer.
var timer = window.setTimeout(simpleMessage, 5000);
window.clearTimeout(timer);

// set interval. will repeat every  5000ms
window.setInterval(simpleMessage, 5000);

// if you wanted to clear the intervals.

var intervalHandler = window.setInterval(simpleMessage, 5000);
window.clearInterval(intervalHandle);
```

Type Checking

```
var myNumber = 1;
var myString = 'some Text';
var bools = true;
var myArray = [];
var myObj = {};
var notNumber = NaN;
var nullified = null;
var undef;

typeof myNumber;
// returns 'number'

typeof myString;
// returns 'string'

typeof bools;
// returns 'boolean'

typeof myArray;
// returns 'object'.

// Not super helpful so must check if it has length property to see if it is an array.
typeof myArray === 'object' && myArray.hasOwnProperty('length');
// returns true

typeof myObj;
// returns 'object'. Must do the same test as above but expect false back from check.

typeof notNumber;
```

```
// returns 'number'. this is confusing but returns this as NaN is part of the global Number object.

// must check if isNaN()
typeof notNumber === 'number' && isNaN(notNumber);
// returns true if type of is 'number' and is still NaN

typeof undef;
// returns 'undefined'

undef === undefined && typeof undef === 'undefined';
// returns 'true'

notDeclared === undefined;
// -> Uncaught ReferenceError: notDeclared is not defined
```

Add default arguments to a function

```
var myFunc = function (arg1='default argument one', arg2='default argument two') {
  console.log(arg1 + " & " + arg2);
};

myFunc(undefined, 'and a new value'); // logs 'default argument one & and a new value'
```

Throttle or Debounce Functions Calls

```
var helpers = {
  /**
   * debouncing, executes the function if there was no new event in $wait milliseconds
   * @param func
   * @param wait
   * @param scope
   * @returns {Function}
   */
  debounce: function(func, wait, scope) {
    var timeout;
    return function() {
      var context = scope || this, args = arguments;
      var later = function() {
        timeout = null;
        func.apply(context, args);
      };
      clearTimeout(timeout);
      timeout = setTimeout(later, wait);
    };
  },

  /**
   * In case of a "storm of events", this executes once every $threshold
   * @param fn
   * @param threshold
   * @param scope
   * @returns {Function}
   */
  throttle: function(fn, threshold, scope) {
    threshold || (threshold = 250);
    var last, deferTimer;

    return function() {
      var context = scope || this;
      var now = +new Date, args = arguments;

      if (last && now < last + threshold) {
        // Hold on to it
        clearTimeout(deferTimer);
        deferTimer = setTimeout(function() {
          last = now;
          fn.apply(context, args);
        }, threshold);
      } else {
        last = now;
        fn.apply(context, args);
```