Byte-saving Techniques

Byte-saving Techniques

This is a collection of JavaScript wizardry that can shave bytes off of your code. It's mainly intended as a reference for those creating entries for 140byt.es. Feel free to add your own or send any feedback to @140bytes.

Disclaimer

Outside of the 140bytes challenge or other code golf challenges, please be considerate and don't pre-minify code you wish to share with others. We have minifiers for that.

Arguments

Use one-letter positional arguments, in alphabetical order

Since arguments will need to be as short as possible, and will likely be reused within their lifetime, it's best to treat them as positionals instead of trying to give them meaning through their name. While using one-letter names marginally aids readability for a single function, keeping a consistent approach helps readability across all functions.

```
function(t,d,v,i,f){...} // before
function(a,b,c,d,e){...} // after
```

Test argument presence instead of length

Use `in` to check whether a given argument was passed

```
arguments.length>1||(cb=alert) // before
1 in arguments||(cb=alert)     // after
```

Variables

Use placeholder arguments instead of `var`

Save bytes on the `var` declaration by putting placeholder arguments in the function declaration.

```
function(a){var b=1;...} // before
function(a,b){b=1;...}   // after
```

Re-use variables where possible

Careful reuse of a variable that is no longer needed can save bytes.

```
setTimeout(function(){for(var i=10;i--;)... }, a) // before
setTimeout(function(){for(a=10;a--;)... }, a)     // after
```

Assign wherever possible

Since assignment returns the assigned value, perform assignment and evaluation at the same time to save bytes. A good example of this is @jed's JSONP function, where the string `script` is assigned in the `createElement` method.

```
a=this.localStorage;if(a){...} // before
if(a=this.localStorage){...}   // after
```

Use an array to swap variables

An array can be used as a temporary placeholder to avoid declaring another variable.

```
var a=1,b=2,c;c=a;a=b;b=c // before
var a=1,b=2;a=[b,b=a][0]  // after
```

Exploit coercion

JavaScript coercion is a blessing and a curse, but sometimes it can be very useful. @jed's pubsub function decrements a negative variable, and then concatenates the results with a string, resulting in a string like someString-123, which is exploited later by using the hyphen as a split token to return the original string.

Loops

Omit loop bodies

If you can perform all the logic you need within the conditional part of a loop, you don't need the loop body. For an example, see @jed's timeAgo function.

Use `for` over `while`

`for` and `while` require the same number of bytes, but `for` gives you more control and assignment opportunity.

```
while(i--){...} // before
for(;i--;){...} // after

i=10;while(i--){...} // before
for(i=10;i--;){...}  // after
```

Use index presence to iterate arrays of truthy items

When iterating over an array of objects that you know are truthy, short circuit on object presence to save bytes.

```
for(a=[1,2,3,4,5],l=a.length,i=0;i<l;i++){b=a[i];...}
for(a=[1,2,3,4,5],i=0;b=a[i++];){...}
```

Use `for..in` with assignment to get the keys of an object

```
a=[];for(b in window)a.push(window[b]) // before
a=[];i=0;for(a[i++]in window);        // after
```

Operators

Understand operator precedence

This Mozilla page is an excellent resource to get started.

Understand bitwise operator hacks

Use ~ with indexOf to test presence

```
hasAnF="This sentence has an f.".indexOf("f")>=0 // before
hasAnF=~"This sentence has an f.".indexOf("f")   // after
```

Use `,` to chain expressions on one conditional line

```
with(document){open();write("hello");close()}
with(document)open(),write("hello"),close()
```

Use `[]._` instead of `void 0`, instead of `undefined` (`""._`, `1.._` and `0[0]` also work, but are slower)

Remove unnecessary space after an operator

Whitespace isn't always needed after an operator, and sometimes may be omitted:

```
typeof [] // before
typeof[]  // after
```

Numbers

Use `~~` and `0|` instead of `Math.floor`

Both of these operator combos will floor numbers (note that since `~` has lower precedence than `|`, they are not identical).

```
rand10=Math.floor(Math.random()*10) // before
rand10=0|Math.random()*10           // after
```

Use `AeB` format for large denary numbers

This is equivalent to `A*Math.pow(10,B)`.

```
million=1000000 // before
million=1e6     // after
```

Use `A<<B` format for large binary numbers

This is equivalent to `A*Math.pow(2,B)`. See @jed's rgb2hex for an example.

```
color=0x100000 // before
color=1<<20    // after
```

Use `1/0` instead of `Infinity`

It’s shorter. Besides, division by zero gets you free internet points.

```
[Infinity,-Infinity] // before
[1/0,-1/0]           // after
```

Exploit the "falsiness" of 0

When comparing numbers, it's often shorter to munge the value to 0 first.

```
a==1||console.log("not one") // before
~-a||console.log("not one")  // after
```

Use `~` to coerce any non-number to -1,

Used together with the unary `-`, this is a great way to increment numerical variables not yet initialized. This is used on @jed's JSONP implementation.

```
i=i||0;i++ // before
i=-~i      // after
```

Strings

Split using 0

Save two bytes by using a number as a delimiter in a string to be split, as seen in @jed's timeAgo function.

```
"alpha,bravo,charlie".split(",") // before
"alpha0bravo0charlie".split(0)   // after
```

Use the little-known `.link` method

Strings have a built-in `.link` method that creates an HTML link. This is used in @jed's linkify function.

```
html="<a href='"+url+"'>"+text+"</a>" // before
html=text.link(url)                   // after
```

Strings also have several other methods related to HTML, as documented here.

Use `replace` or `.exec` for powerful string iteration

Since the `.replace` method can take a function as its second argument, it can handle a lot of iteration bookkeeping for you. You can see this exploited in @jed's templates and UUID functions.

Use `Array` to repeat a string

```
for(a="",i=32;i--;)a+=0 // before
a=Array(33).join(0)     // after
```

Conditionals

Use `&&` and `||` where possible

```
if(a)if(b)return c // before
return a&&b&&c     // after

if(!a)a=Infinity // before
a||(a=Infinity)  // after
```

Arrays

Use elision

Array elision can be used in some cases to save bytes. See @jed's router API for a real-world example.

```
[undefined,undefined,2] // before
[,,2]                   // after

// Note: Be mindful of elided elements at the end of the element list
[2,undefined,undefined] // before length is 3
[2,,]                   // after length is 2
```

Regular Expressions

Denormalize to shorten

While `/\d{2}/` looks smarter, `/\d\d/` is shorter.

`eval` for a regexp literal can be shorter than `RegExp()`

```
r=new RegExp("{"+p+"}","g") // before
r=eval("/{"+p+"}/g")     // after
```

Booleans

Use `!` to create booleans

`true` and `false` can be created by combining the `!` operator with numbers.

```
[true,false] // before
[!0,!1]      // after
```

Functions

Use named functions for recursion

Recursion is often more terse than looping, because it offloads bookkeeping to the stack. This is used in @jed's walk function.

Use named functions for saving state

If state needs to be saved between function calls, name the function and use it as a container. This is used for a counter in @jed's JSONP function.

```
function(i){return function(){console.log("called "+(++i)+" times")}}(0) // before
(function a(){console.log("called "+(a.i=-~a.i)+" times")})            // after
0,function a(){console.log("called "+(a.i=-~a.i)+" times")}            // another alternative
```

Omit `()` on `new` calls w/o arguments

`new Object` is equivalent to `new Object()`

```
now = +new Date() // before
now = +new Date   // after
```

Omit the `new` keyword when possible

Some constructors don't actually require the `new` keyword.

```
r=new Regexp(".",g) // before
r=Regexp(".",g)     // after

l=new Function("x","console.log(x)") // before
l=Function("x","console.log(x)")     // after
```

The `return` statement

When returning anything but a variable, there's no need to use a space after `return`:

```
return ['foo',42,'bar']; // before
return['foo',42,'bar'];  // after
return {x:42,y:417}; // before
return{x:42,y:417};  // after
return .01; // before
return.01;  // after
```

Use the right closure for the job

If you need to execute a function instantly, use the most appropriate closure.

```
;(function(){...})() // before
new function(){...}  // after, if you plan on returning an object and can use `this`
!function(){...}()   // after, if you don't need to return anything
```

In the browser

Use browser objects to avoid logic

Instead of writing your own logic, you can use browser anchor elements to parse URLs as in @jed's parseURL, and text nodes to escape HTML as in @jed's escapeHTML.

Use global scope

Since `window` is the global object in a browser, you can directly reference any property of it. This is well known for things like `document` and `location`, but it's also useful for other properties like `innerWidth`, as shown in @bmiedlar's screensaver.

APIs

Pass static data via argument where possible

Use extra bytes to provide default values

Do one thing and do it well

Other resources

- Ben Alman's explanation of his JS1K entry
- Marijn Haverbeke's explanation of his JS1K entry
- Martin Kleppe's presentation about his JS1K entry
- Suggested Closure Compiler optimizations
- Angus Croll's blog
- Aivo Paas's jscrush

Last edited by badboy, about an hour ago

Delete this Page

Markdown Cheat Sheet

Format Text

Headers

```
# This is an <h1> tag
## This is an <h2> tag
###### This is an <h6> tag
```

Text styles

```
*This text will be italic*
_This will also be italic_
**This text will be bold**
__This will also be bold__

*You **can** combine them*
```

Lists

Unordered

```
* Item 1
* Item 2
  * Item 2a
  * Item 2b
```

Ordered

```
1. Item 1
2. Item 2
3. Item 3
   * Item 3a
   * Item 3b
```

Miscellaneous

Images

```
![GitHub Logo](/images/logo.png)
Format: ![Alt Text](url)
```

Links

```
http://github.com - automatic!
[GitHub](http://github.com)
```

Blockquotes

```
As Kanye West said:
> We're living the future so
> the present is our past.
```

Code Examples in Markdown

Syntax highlighting with GFM

~~~
```javascript
function fancyAlert(arg) {
  if(arg) {
    $.facebox({div:'#foo'})
  }
}
```
~~~

Or, indent your code 4 spaces

```
Here is a Python code example
without syntax highlighting:

    def foo:
      if not bar:
        return true
```

Inline code for comments

```
I think you should use an
`<addr>` element here instead.
```