Cheatography

JavaScript Cheat Sheet

by yjcyun via cheatography.com/165804/cs/34715/

Linking JS file

Inline

> <head><script>const role="developer"</script></head>

Linking

> <body> <script src="script.js"></script></body>

Primitive Data Types

string	const role="dev"
number	const years=10
boolean	const isTrue=true
undefined	let children;
null	let empty=null
symbol	unique value
bigint	large integers that number type can't hold

Truthy and Falsy

falsy values	0, "", undefined, null, NaN
truthy values	everything else

let vs. const vs. var

let	- can be reassigned - block scoped
const	- cannot be reassigned - block scoped - best practice
var	- hoisted(can be referenced before being declared) - can be redeclared - do not use in modern JS

Conditionals

```
// If else statement
if(age >= 18) {
  status = " adu lt"
} else {
  status = " chi ld"
}
```

Conditionals (cont)

```
// Switch statement
switch (day) {
 case " mon day ": // if(day -
=== " mon day ")
      con sol e.l og( "It's
Monday ")
      break; // exits out of the
switch statement
 case " sun day ":
 case " sat urd ay": // if(day -
=== " sat urd ay" || day ===" -
sun day ")
      con sol e.l og( "It's the
weeken d")
      break;
  def ault: // else ...
      con sol e.l og( "Not a
valid day")
// Ternary Operator
const drink = age >= 18 ? " -
bee r" : " bubble tea"
```

Basic Operators

add/subtract	a + b - c
multiply/divide	a * b / 100
power: 2x2x2	2 ** 3
remainder: 13/5 remainder is 3	13 % 5
postfix increment/decrement: returns value before increment	a++, a--
prefix increment/decrement: returns value after increment	++a, --a
find type of variable	typeof a
assignment	a = 'string'
a = a + b	a+=b (*=, /=, -=)
less and greater than	a < b, a > b
less or equal, greater or equal	a <= b, a >= b
logical and, or	a && b, a || b
logical not	!(a === b)

Basic Operators (cont)

strict equal, not equal: no type coercion	a === b, a !== b
loose equal, not equal: type coercion (do not use)	a == b, a != b

Type Conversion and Coercion

convert to number	Number("20-00")
convert to string	String(2000)
convert to boolean	Boolean("hell-o")
coerced to string: "a is 16"	"a is" + 16
coerced to number: 80	"100" - 10 - "-10"
coerced to boolean: if(true) {}	if (hasMoney) {}

Objects

definition	- a data structure where properties are stored in key-value pairs - order does not matter ex. `const student = { firstName: " - She ldo n", lastNa me= " Coo - per ", age: 10, fullName: () => this.f irs tName + ' ' + this.l - astName }`
getting the value	`studen t.f irs - tName` or `studen t["f irs - tNa me"]`
setting the value	`studen t.i sSm - art =true`
object methods	a function attached to an object
calling a method	`studen t.f ull - Name()`

Iteration

for Loop

```
// loops forward
for (let i = 1; i <= 10; i++) {
    con sol e.l og(i)
}
// loops backward
for (let i = 10; i > 0; i--) {
    con sol e.l og(i)
}
```

break statement: completely terminates the loop

```
for (let i = 1; i <= 10; i++) {
   if(i === 3) break;
    con sol e.l og(i); // 1, 2
}
```

continue statement: terminates the current iteration

```
for (let i = 1; i <= 10; i++) {
   if(i === 3) continue;
    con sol e.l og(i); // 1, 2,
4, 5, 6, 7, 8, 9, 10
}
```

while loop

```
let i = 1; // initialize counter
while(i <= 10) { // condition
  con sol e.l og(i);
 i++; // update counter
}
```

Functions

purpose

- a block of code that performs certain tasks
- useful for repeated tasks
- either does something or returns value(s)

parameters: variable(s) functions accept

```
function fullNa me( first,
last, ...params) {}
```

arguments: values passed in when function is invoked

```
fullNa me( " She ldo n", " -
Coo per ", ...arg uments)
```

Functions (cont)

function declaration

- define function with a function name
- can be invoked before they are defined

ex. `function foo(bar) {}`

function expression

- anonymous function stored in a variable
- cannot be invoked before they are defined

ex. `const foo = function (bar) {}`

arrow functions

- has no `this` keyword

ex. `const foo = bar => bar + a`

default parameters

- allow params to be initialized with default values if no value/`undefined` is passed

ex. `function foo (bar = 0) {}`

primitive type argument

- function makes a copy of the original value

reference type argument

- functions makes a copy of the reference (the copy is still a value)
- JS, arguments can only be passed by value

first-class function

a concept in programming where functions are treated as first-class citizens meaning they can behave like variables

higher-order function

- a function that can be passed as an argument to other functions, or one that can be returned by another function or do both

ex. `bar.ad dEv ent Lis ten - er( 'cl ick', foo)`

ex2. `const foo = () => () => consol e.l og( 'he llo')`

Functions (cont)

.call(object, arg1, arg2, ...)

- calls the function with a given `this` value and arguments provided individually
- remember `this` is undefined on regular function call

ex.
```
foo(nu m,str) {return
this.name + str}
foo(12, " hel lo") // 'this'
keyword is undefined
foo.ca ll( bar Object, 12, " -
hel lo")
```

.apply (ob ject, [arg1, arg2, ...])

- calls the function with a given `this` value, and arguments provided as an array

ex.
```
const arguments = [12, " -
hel lo"]
foo.ap ply (ba rOb ject,
arguments)
foo.ca ll( bar Object,
...arg uments)
```

.bind(object, arg?)

- returns a new function with `this` value bound to the provided object
- arguments can be passed optionally to preset arguments

ex. `const newFoo = foo.bi nd( - bar Object)`

IIFE

- immediately invoked function expression
- useful if the function is only to be used once

ex. `(funct ion() {conso le.l - og ('h ell o')})()`

closure

- a function that remembers all the variables that existed at the function's birth place