

n Number() = 42

PROPERTIES

- n**.POSITIVE_INFINITY +∞ equivalent
- n**.NEGATIVE_INFINITY -∞ equivalent
- n**.MAX_VALUE largest positive value
- n**.MIN_VALUE smallest positive value
- n**.EPSILON diff between 1 & smallest >1
- n**.NaN not-a-number value

METHODS

- s**.toExponential(**dec**) exp. notation
- s**.toFixed(**dec**) fixed-point notation
- s**.toPrecision(**p**) change precision
- b**.isFinite(**n**) check if number is finite
- b**.isInteger(**n**) check if number is int.
- b**.isNaN(**n**) check if number is NaN
- n**.parseInt(**s**, **radix**) string to integer
- n**.parseFloat(**s**, **radix**) string to float

r Regexp() = /.+/ig

PROPERTIES

- n**.lastIndex index to start global regexp
- s**.flags active flags of current regexp
- b**.global flag g (search all matches)
- b**.ignoreCase flag i (match lower/upper)
- b**.multiline flag m (match multiple lines)
- b**.sticky flag y (search from lastIndex)
- b**.unicode flag u (enable unicode feat.)
- s**.source current regexp (w/o slashes)

METHODS

- a**.exec(**str**) exec search for a match
- b**.test(**str**) check if regexp match w/str

CLASSES

- . any character \t tabulator
- \d digit [0-9] \r carriage return
- \D no digit [^0-9] \n line feed
- \w any alphanumeric char [A-Za-z0-9_]
- \W no alphanumeric char [^A-Za-z0-9_]
- \s any space char (space, tab, enter...)
- \S no space char (space, tab, enter...)
- \xN char with code N [b] backspace
- \uN char with unicode N \0 NUL char

CHARACTER SETS OR ALTERNATION

- [abc] match any character set
- ^[abc] match any char. set not enclosed
- a|b match a or b

BOUNDARIES

- ^ begin of input \$ end of input
- \b zero-width word boundary
- \B zero-width non-word boundary

GROUPING

- (x) capture group (?x) no capture group
- \n reference to group n captured

QUANTIFIERS

- x* preceding x 0 or more times {0,}
- x+ preceding x 1 or more times {1,}
- x? preceding x 0 or 1 times {0,1}
- x{n} n occurrences of x
- x{n,} at least n occurrences of x
- x{n,m} between n & m occurrences of x

ASSERTIONS

- x(=?y) x (only if x is followed by y)
- x(?!y) x (only if x is not followed by y)

s String() = 'text'

PROPERTIES

- n**.length string size

METHODS

- s**.charAt(**index**) char at position [i]
- n**.charCodeAt(**index**) unicode at pos.
- s**.fromCharCode(**n1**, **n2**...) code to char
- s**.concat(**str1**, **str2**...) combine text +
- b**.startsWith(**str**, **size**) check beginning
- b**.endsWith(**str**, **size**) check ending
- b**.includes(**str**, **from**) include substring?
- n**.indexOf(**str**, **from**) find substr index
- n**.lastIndexOf(**str**, **from**) find from end
- n**.search(**regex**) search & return index
- n**.localeCompare(**str**, **locale**, **options**)
- a**.match(**regex**) matches against string
- s**.repeat(**n**) repeat string n times
- s**.replace(**str**|**regex**, **newstr**|**func**)
- s**.slice(**ini**, **end**) str between ini/end
- s**.substr(**ini**, **len**) substr of len length
- s**.substring(**ini**, **end**) substr fragment
- a**.split(**sepl**|**regex**, **limit**) divide string
- s**.toLowerCase() string to lowercase
- s**.toUpperCase() string to uppercase
- s**.trim() remove space from begin/end
- s**.raw`` template strings with \${vars}

d Date()

METHODS

- n**.UTC(**y**, **m**, **d**, **h**, **i**, **s**, **ms**) timestamp
- n**.now() timestamp of current time
- n**.parse(**str**) convert str to timestamp
- n**.setTime(**ts**) set UNIX timestamp
- n**.getTime() return UNIX timestamp

UNIT SETTERS (ALSO .setUTC*() methods)

- n**.setFullYear(**y**, **m**, **d**) set year (yyyy)
- n**.setMonth(**m**, **d**) set month (0-11)
- n**.setDate(**d**) set day (1-31)
- n**.setHours(**h**, **m**, **s**, **ms**) set hour (0-23)
- n**.setMinutes(**m**, **s**, **ms**) set min (0-59)
- n**.setSeconds(**s**, **ms**) set sec (0-59)
- n**.setMilliseconds(**ms**) set ms (0-999)

UNIT GETTERS (ALSO .getUTC*() methods)

- n**.getDate() return day (1-31)
- n**.getDay() return day of week (0-6)
- n**.getMonth() return month (0-11)
- n**.getFullYear() return year (yyyy)
- n**.getHours() return hour (0-23)
- n**.getMinutes() return minutes (0-59)
- n**.getSeconds() return seconds (0-59)
- n**.getMilliseconds() return ms (0-999)

LOCALE & TIMEZONE METHODS

- n**.getTimezoneOffset() offset in mins
- s**.toLocaleDateString(**locale**, **options**)
- s**.toLocaleTimeString(**locale**, **options**)
- s**.toLocaleString(**locale**, **options**)
- s**.toUTCString() return UTC date
- s**.toDateString() return American date
- s**.toTimeString() return American time
- s**.toISOString() return ISO8601 date
- s**.toJSON() return date ready for JSON

a Array() = [1, 2, 3]

PROPERTIES

- n**.length number of elements

METHODS

- b**.isArray(**obj**) check if obj is array
- b**.includes(**obj**, **from**) include element?
- n**.indexOf(**obj**, **from**) find elem. index
- n**.lastIndexOf(**obj**, **from**) find from end
- s**.join(**sep**) join elements w/separator
- a**.slice(**ini**, **end**) return array portion
- a**.concat(**obj1**, **obj2**...) return joined array

MODIFY SOURCE ARRAY METHODS

- a**.copyWithin(**pos**, **ini**, **end**) copy elems
- a**.fill(**obj**, **ini**, **end**) fill array with obj
- a**.reverse() reverse array & return it
- a**.sort(**cf(a,b)**) sort array (unicode sort)
- a**.splice(**ini**, **del**, **o1**, **o2**...) del&add elem

ITERATION METHODS

- a**.entries() iterate key/value pair array
- a**.keys() iterate only keys array
- a**.values() iterate only values array

CALLBACK FOR EACH METHODS

- b**.every(**cb(e,i,a)**, **arg**) test until false
- b**.some(**cb(e,i,a)**, **arg**) test until true
- a**.map(**cb(e,i,a)**, **arg**) make array
- a**.filter(**cb(e,i,a)**, **arg**) make array w/true
- o**.find(**cb(e,i,a)**, **arg**) return elem w/true
- n**.findIndex(**cb(e,i,a)**, **arg**) return index
- o**.forEach(**cb(e,i,a)**, **arg**) exec for each
- o**.reduce(**cb(p,e,i,a)**, **arg**) accumulative
- o**.reduceRight(**cb(p,e,i,a)**, **arg**) from end

ADD/REMOVE METHODS

- o**.pop() remove & return last element
- n**.push(**o1**, **o2**...) add element & return length
- o**.shift() remove & return first element
- n**.unshift(**o1**, **o2**...) add element & return len

b Boolean() = true / false

no own properties or methods

f Function() = function(a, b) { ... }

PROPERTIES

- o**.length return number of arguments
- s**.name return name of function
- o**.prototype prototype object

METHODS

- o**.call(**newthis**, **arg1**, **arg2**...) change this
- o**.apply(**newthis**, **arg1**) with args array
- o**.bind(**newthis**, **arg1**, **arg2**...) bound func

- n** number
- NaN** (not-a-number)
- s** string
- b** boolean (true/false)
- a** array
- d** date
- r** regular expression
- f** function
- o** object
- undefined**

only available on ECMAScript 6

- n** static (ex: Math.random())
- n** non-static (ex: new Date().getDate())
- argument** required
- argument** optional

Math

PROPERTIES

- n**.E Euler's constant
- n**.LN2 natural logarithm of 2
- n**.LN10 natural logarithm of 10
- n**.LOG2E base 2 logarithm of E
- n**.LOG10E base 10 logarithm of E
- n**.PI ratio circumference/diameter
- n**.SQRT1_2 square root of 1/2
- n**.SQRT2 square root of 2

METHODS

- n**.abs(**x**) absolute value
- n**.cbrt(**x**) cube root
- n**.clz32(**x**) return leading zero bits (32)
- n**.exp(**x**) return e^x
- n**.expm1(**x**) return $e^x - 1$
- n**.hypot(**x1**, **x2**...) length of hypotenuse
- n**.imul(**a**, **b**) signed multiply
- n**.log(**x**) natural logarithm (base e)
- n**.log1p(**x**) natural logarithm (1+x)
- n**.log10(**x**) base 10 logarithm
- n**.log2(**x**) base 2 logarithm
- n**.max(**x1**, **x2**...) return max number
- n**.min(**x1**, **x2**...) return min number
- n**.pow(**base**, **exp**) return $base^{exp}$
- n**.random() float random number [0,1)
- n**.sign(**x**) return sign of number
- n**.sqrt(**x**) square root of number

ROUND METHODS

- n**.ceil(**x**) superior round (smallest)
- n**.floor(**x**) inferior round (largest)
- n**.fround(**x**) nearest single precision
- n**.round(**x**) round (nearest integer)
- n**.trunc(**x**) remove fractional digits

TRIGONOMETRIC METHODS

- n**.acos(**x**) arccosine
- n**.acosh(**x**) hyperbolic arccosine
- n**.asin(**x**) arcsine
- n**.asinh(**x**) hyperbolic arcsine
- n**.atan(**x**) arctangent
- n**.atan2(**x**, **y**) arctangent of quotient x/y
- n**.atanh(**x**) hyperbolic arctangent
- n**.cos(**x**) cosine
- n**.cosh(**x**) hyperbolic cosine
- n**.sin(**x**) sine
- n**.sinh(**x**) hyperbolic sine
- n**.tan(**x**) tangent
- n**.tanh(**x**) hyperbolic tangent

JSON

METHODS

- n**.parse(**str**, **tf**(**k**,**v**)) parse string to object
- n**.stringify(**obj**, **repf**[**wl**, **sp**]) convert to str

Error()

PROPERTIES

- s**.name return name of error
- s**.message return description of error

EvalError(), InternalError(), RangeError(), URIError(), ReferenceError(), SyntaxError(), TypeError()

Object() = {key: value, key2: value2}

PROPERTIES

- o**.constructor return ref. to object func.

METHODS

- o**.assign(**dst**, **src1**, **src2**...) copy values
- o**.create(**proto**, **prop**) create obj w/prop
- o**.defineProperties(**obj**, **prop**)
- o**.defineProperty(**obj**, **prop**, **desc**)
- o**.freeze(**obj**) avoid properties changes
- o**.getOwnPropertyDescriptor(**obj**, **prop**)
- a**.getOwnPropertyNames(**obj**)
- a**.getOwnPropertySymbols(**obj**)
- o**.getPrototypeOf(**obj**) return prototype
- b**.is(**val1**, **val2**) check if are same value
- b**.isExtensible(**obj**) check if can add prop
- b**.isFrozen(**obj**) check if obj is frozen
- b**.isSealed(**obj**) check if obj is sealed
- a**.keys(**obj**) return only keys of object
- o**.preventExtensions(**obj**) avoid extend
- o**.seal(**obj**) prop are non-configurable
- o**.setPrototypeOf(**obj**, **prot**) change prot

INSTANCE METHODS

- b**.hasOwnProperty(**prop**) check if exist
- b**.isPrototypeOf(**obj**) test in another obj
- b**.propertyIsEnumerable(**prop**)
- s**.toString() return equivalent string
- s**.toLocaleString() return locale version
- o**.valueOf() return primitive value

Promise()

METHODS

- p**.all(**obj**) return promise
- p**.catch(**onRejected**(**s**)) = .then(**undef**,**s**)
- p**.then(**onFulfilled**(**v**), **onRejected**(**s**))
- p**.race(**obj**) return greedy promise (res/rej)
- p**.resolve(**obj**) return resolved promise
- p**.reject(**reason**) return rejected promise

p Proxy() Reflect same methods (not func)

METHODS

- o**.apply(**obj**, **arg**, **arglist**) trap function call
- o**.construct(**obj**, **arglist**) trap new oper
- o**.defineProperty(**obj**, **prop**, **desc**)
- o**.deleteProperty(**obj**, **prop**) trap delete
- o**.enumerate(**obj**) trap for...in
- o**.get(**obj**, **prop**, **rec**) trap get property
- o**.getOwnPropertyDescriptor(**obj**, **prop**)
- o**.getPrototypeOf(**obj**)
- o**.has(**obj**, **prop**) trap in operator
- o**.ownKeys(**obj**)
- o**.preventExtensions(**obj**)
- o**.set(**obj**, **prop**, **value**) trap set property
- o**.setPrototypeOf(**obj**, **proto**)

globals

METHODS

- o** eval(**str**) evaluate javascript code
- b** isFinite(**obj**) check if is a finite number
- b** isNaN(**obj**) check if is not a number
- n** parseInt(**s**, **radix**) string to integer
- n** parseFloat(**s**, **radix**) string to float
- s** encodeURIComponent(**URI**) = to %3D
- s** decodeURIComponent(**URI**) %3D to =

s Set()

WeakSet only obj as items

PROPERTIES

- n**.size return number of items

METHODS

- s**.add(**item**) add item to set **ws**
- b**.has(**item**) check if item exists **ws**
- b**.delete(**item**) del item & return if del **ws**
- o**.clear() remove all items from set

ITERATION METHODS

- si**.entries() iterate items
- si**.values() iterate only value of items

CALLBACK FOR EACH METHODS

- o**.forEach(**cb**(**e**,**i**,**a**), **arg**) exec for each

m Map()

WeakMap only obj as keys

PROPERTIES

- n**.size return number of elements

METHODS

- m**.set(**key**, **value**) add pair key=value **wm**
- o**.get(**key**) return value of key **wm**
- b**.has(**key**) check if key exist **wm**
- b**.delete(**key**) del elem. & return if ok **wm**
- o**.clear() remove all elements from map

ITERATION METHODS

- m**.entries() iterate elements
- m**.keys() iterate only keys
- m**.values() iterate only values

CALLBACK FOR EACH METHODS

- o**.forEach(**cb**(**e**,**i**,**a**), **arg**) exec for each

Symbol()

PROPERTIES

- s**.iterator specifies default iterator
- s**.match specifies match of regexp
- s**.species specifies constructor function

METHODS

- s**.for(**key**) search existing symbols
- s**.keyFor(**sym**) return key from global reg

g Generator() = function* () { ... }

METHODS

- o**.next(**value**) return obj w/{value,done}
- o**.return(**value**) return value & true done
- o**.throw(**except**) throw an error

Others

FAST TIPS

- var declare variable
- let declare block scope local variable
- const declare constant (read-only)
- func(**a**=1) default parameter value
- func(...**a**) rest argument (spread operator)
- (**a**) => { ... } function equivalent (fat arrow)
- `string \${a}` template with variables
- 0**n** binary (2) number **n** to decimal
- 0**o**n octal (8) number **n** to decimal
- 0**x**n hexadecimal (16) number **n** to decimal
- for (**i** in **array**) { ... } iterate array, **i** = index
- for (**e** of **array**) { ... } iterate array, **e** = value
- class **B** extends **A** { } class sugar syntax