11-dars. JS sonlar va matn ustida amallar (Numbers, Strings)

11-dars. JS sonlar va matnlar ustida amallar (Numbers, Strings)

Number

- int
- float

-9,007,199,254,740,992 (-2^53) and 9,007,199,254,740,992 (2^53)

Integer

```
0
3
10000000
```

Floating-Point Literals

✎ The syntax is:

```
[digits][.digits][(E|e)[(+|-)]digits]
```

```
3.14
2345.6789
.333333333333333333
6.02e23 // 6.02 × 10²³
1.4738223E-32 //
1.4738223 × 10⁻³²
```

✎ Separators

```
let billion = 1_000_000_000; // Underscore as a thousands separator.
```

Arithmetic in JavaScript

- -

```
let result = 10 + 20 // 30
```

- -

```
let result = 10 - 20 // -10
```

- /

```
let result  = 10 / 20 // 0.5
```

- -

```
let result  = 10 * 20 // 200
```

- %

```
let result  = 5 % 2 // 1
```

- **

```
let result  = 5 * 2 // 25
```

Math object:

> *JavaScript supports more complex mathematical operations through a set of functions and constants defined as properties of the Math object:*

```
Math.pow(2,53) // ⇒ 9007199254740992: 2 to the power 53
Math.round(.6) // ⇒ 1.0: round to the nearest integer
Math.ceil(.6) // ⇒ 1.0: round up to an integer
Math.floor(.6) // ⇒ 0.0: round down to an integer
Math.abs(-5) // ⇒ 5: absolute value
Math.max(x,y,z) // Return the largest argument
```

```
Math.min(x,y,z) // Return the smallest argument
Math.random() // Pseudo-random number x where 0 ⩽ x < 1.0 Math.PI // π: circumference of a circle / diameter
Math.E // e: The base of the natural logarithm
Math.sqrt(3) // ⇒ 3**0.5: the square root of 3
Math.pow(3, 1/3) // ⇒ 3**(1/3): the cube root of 3
Math.sin(0) // Trigonometry: also
Math.cos,
Math.atan, etc.
Math.log(10) // Natural logarithm of 10
Math.log(100)/Math.LN10 // Base 10 logarithm of 100
Math.log(512)/Math.LN2 // Base 2 logarithm of 512
Math.exp(3) // Math.E cubed
```

ES6 defines more functions on the Math object:

```
Math.cbrt(27) // ⇒ 3: cube root
Math.hypot(3, 4) // ⇒ 5: square root of sum of squares of all arguments
Math.log10(100) // ⇒ 2: Base-10 logarithm
Math.log2(1024) // ⇒ 10: Base-2 logarithm
Math.log1p(x) // Natural log of (1+x); accurate for very small x
Math.expm1(x) // Math.exp(x)-1; the inverse of Math.log1p()
Math.sign(x) // -1, 0, or 1 for arguments <, ==, or > 0
Math.imul(2,3) // ⇒ 6: optimized multiplication of 32-bit integers
Math.clz32(0xf) // ⇒ 28: number of leading zero bits in a 32-bit integer
Math.trunc(3.9) // ⇒ 3: convert to an integer by truncating fractional part
Math.fround(x) // Round to nearest 32-bit float number
Math.sinh(x) // Hyperbolic sine. Also Math.cosh(), Math.tanh()
Math.asinh(x) // Hyperbolic arcsine. Also Math.acosh(), Math.atanh()
```

```
Infinity // A positive number too big to represent
Number.POSITIVE_INFINITY // Same value
1/0 // ⇒ Infinity
Number.MAX_VALUE * 2 // ⇒ Infinity; overflow

-Infinity // A negative number too big to represent
Number.NEGATIVE_INFINITY // The same value
-1/0 // ⇒ -Infinity
-Number.MAX_VALUE * 2 // ⇒ -Infinity
```

```
NaN // The not-a-number value
Number.NaN // The same value, written another way
0/0 // ⇒ NaN
Infinity/Infinity // ⇒ NaN
Number.MIN_VALUE/2 // ⇒ 0: underflow -
Number.MIN_VALUE/2 // ⇒ -0: negative zero -
1/Infinity // → -0: also negative 0
-0

// The following Number properties are defined in
ES6 Number.parseInt() // Same as the global parseInt() function
Number.parseFloat() // Same as the global parseFloat() function
Number.isNaN(x) // Is x the NaN value?
Number.isFinite(x) // Is x a number and finite?
Number.isInteger(x) // Is x an integer?
Number.isSafeInteger(x) // Is x an integer -(2**53) < x < 2**53?
Number.MIN_SAFE_INTEGER // ⇒ -(2**53 - 1)
Number.MAX_SAFE_INTEGER // ⇒ 2**53 - 1
Number.EPSILON // ⇒ 2**-52: smallest difference between numbers
```

```
let zero = 0; // Regular zero
let negz = -0; // Negative zero
zero === negz // ⇒ true: zero and negative zero are equal
1/zero === 1/negz // ⇒ false: Infinity and -Infinity are not equal
```

```
let x = .3 - .2; // thirty cents minus 20 cents
let y = .2 - .1; // twenty cents minus 10 cents
x === y // ⇒ false: the two values are not the same!
x === .1 // ⇒ false: .3-.2 is not equal to .1
y === .1
```

String

- '
- "
- `

```
"" // The empty string: it has zero characters
'testing' "3.14"
'name="myform"'
"Wouldn't you prefer O'Reilly's book?"
"τ is the ratio of a circle's circumference to its radius"
`"She said 'hi'", he said.`
```

```
// A string representing 2 lines written on one line:
'two\nlines' // A one-line string written on 3 lines:
"one\
long\
line" // A two-line string written on two lines:
`the newline character at the end of this line is included literally in this string`
```

```
'You\'re right, it can\'t be a quote'
```

Table 3-1. JavaScript escape sequences

Sequence	Character represented
`\0`	The NUL character (`\u0000`)
`\b`	Backspace (`\u0008`)
`\t`	Horizontal tab (`\u0009`)
`\n`	Newline (`\u000A`)
`\v`	Vertical tab (`\u000B`)
`\f`	Form feed (`\u000C`)
`\r`	Carriage return (`\u000D`)
`\"`	Double quote (`\u0022`)
`\'`	Apostrophe or single quote (`\u0027`)
`\\`	Backslash (`\u005C`)
`\x`*nn*	The Unicode character specified by the two hexadecimal digits *nn*
`\u`*nnnn*	The Unicode character specified by the four hexadecimal digits *nnnn*
`\u{`*n*`}`	The Unicode character specified by the codepoint *n*, where *n* is one to six hexadecimal digits between 0 and 10FFFF (ES6)

```
let msg = "Hello, " + "world"; // Produces the string "Hello, world"
let greeting = "Welcome to my blog," + " " + name;
```

- `= = =`
- `= =`
- `! =`
- `! = =`

```
s.length
```

JavaScript provides a rich API for working with strings:

```
let s = "Hello, world"; // Start with some text.

// Obtaining portions of a string
s.substring(1,4) // ⇒ "ell": the 2nd, 3rd, and 4th characters.
s.slice(1,4) // ⇒ "ell": same thing
s.slice(-3) // ⇒ "rld": last 3 characters
s.split(", ") // ⇒ ["Hello", "world"]: split at delimiter string

// Searching a string
s.indexOf("l") // ⇒ 2: position of first letter l
s.indexOf("l", 3) // ⇒ 3: position of first "l" at or after 3
s.indexOf("zz") // ⇒ -1: s does not include the substring "zz"
s.lastIndexOf("l") // ⇒ 10: position of last letter l

// Boolean searching functions in ES6 and later
s.startsWith("Hell") // ⇒ true: the string starts with these
s.endsWith("!") // ⇒ false: s does not end with that
s.includes("or") // ⇒ true: s includes substring "or"

// Creating modified versions of a string
s.replace("llo", "ya") // ⇒ "Heya, world" s.toLowerCase() // ⇒ "hello, world"
s.toUpperCase() // ⇒ "HELLO, WORLD"
s.normalize() // Unicode NFC normalization: ES6
s.normalize("NFD") // NFD normalization. Also "NFKC", "NFKD"

// Inspecting individual (16-bit) characters of a string
s.charAt(0) // ⇒ "H": the first character
s.charAt(s.length-1) // ⇒ "d": the last character
s.charCodeAt(0) // ⇒ 72: 16-bit number at the specified position
s.codePointAt(0) // ⇒ 72: ES6, works for codepoints > 16 bits

//String padding functions in ES2017
"x".padStart(3) // ⇒ " x": add spaces on the left to a length of 3
"x".padEnd(3) // ⇒ "x ": add spaces on the right to a length of 3
```

```
"x".padStart(3, "*") // ⇒ "**x": add stars on the left to a length of 3
"x".padEnd(3, "-") // ⇒ "x--": add dashes on the right to a length of 3

// Space trimming functions. trim() is ES5; others ES2019
" test ".trim() // ⇒ "test": remove spaces at start and end
" test ".trimStart() // ⇒ "test ": remove spaces on left. Also trimLeft
" test ".trimEnd() // ⇒ " test": remove spaces at right. Also trimRight // Miscellaneous string methods
s.concat("!") // ⇒ "Hello, world!": just use + operator instead
"<>".repeat(5) // ⇒ "<><><><><>": concatenate n copies. ES6
```

> *Remember that strings are immutable in JavaScript. Methods like replace() and toUpperCase() return new strings: they do not modify the string on which they are invoked.*

```
let s = "hello, world";
s[0] // ⇒ "h"
s[s.length-1] // ⇒ "d"
```

Template Literals

```
let s = `hello world`;
```

Syntax

```
` ${}`
```

```
let name = "Bill";
let greeting = `Hello ${ name }.`; // greeting == "Hello Bill."
```

```
let errorMessage = `\ \u2718 Test failure at ${filename}:${linenumber}: ${exception.message} Stack trace: ${exception.stack} `;
```

```
function sum(a, b) {
        return a + b;
}

console.log(`1 + 2 = ${sum(1, 2)}.`); // 1 + 2 = 3.
```

```
let guestList = `Guests:
* John
* Pete
* Mary
`;

console.log(guestList); // a list of guests, multiple lines
```

```
let guestList ="Guests:// Error: Unexpected token ILLEGAL
* John"
```

`String.fromCodePoint(code)`

> *Creates a character by its numeric* `code`

```
let str = '';

for (let i = 65; i ⩽ 220; i++) {
        str += String.fromCodePoint(i);
}

console.log( str );
// Output:
// ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~□□□□□□

// ¡¢£¤¥¦§¨©ª«¬®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜ
```

Pattern Matching

> *JavaScript defines a datatype known as a regular expression (or RegExp) for describing and matching patterns in strings of text*

```
/^HTML/; // Match the letters H T M L at the start of a string
/[1-9][0-9]*/; // Match a nonzero digit, followed by any # of digits
/\bjavascript\b/i; // Match "javascript" as a word, caseinsensitive
```

```
let text = "testing: 1, 2, 3"; // Sample text
let pattern = /\d+/g; // Matches all instances of one or more digits
pattern.test(text) // ⇒ true: a match exists
text.search(pattern) // ⇒ 9: position of first match
text.match(pattern) // ⇒ ["1", "2", "3"]: array of all matches
text.replace(pattern, "#") // ⇒ "testing: #, #, #"
text.split(/\D+/) // ⇒ ["","1","2","3"]: split on nondigits
```