All Fundamentals About JavaScript

Born of this Unique Piece of Language

JavaScript was created in 1995 by Brendan Eich while he was an engineer at Netscape.JavaScript was first released with Netscape 2 early in 1996. It was originally going to be called LiveScript.

Several months later, Microsoft released JScript with Internet Explorer 3. It was a mostly-compatible JavaScript work-alike.

Several months after that, Netscape submitted JavaScript to Ecma International, a European standards organization, which resulted in the first edition of the ECMAScript standard that year. The standard received a significant update as ECMAScript edition 3 in 1999, and it has stayed pretty much stable ever since. The fourth edition was abandoned, due to political differences concerning language complexity. Many parts of the fourth edition formed the basis for ECMAScript edition 5, published in December of 2009, and for the 6th major edition of the standard, published in June of 2015.

What is JavaScript?

1.JavaScript is a multi-paradigm , dynamic language with types and operators,standard built-in-objects and methods.

2. Its syntax is based on the Java and C language.

3. JavaScript supports object-oriented programming with object prototypes, instead of classes.

4. JavaScript also support functional programming.

Data types of JavaScript

- Number
- String
- Boolean
- Symbol(start it's journey in ECMA version 6)
- Object
  - Function
  - Array
  - Date
  - Regular Expression(RegExp)
- null
- undefined
- Error

Primitive data types:

- string
- number
- bigint
- boolean
- undefined

- symbol,
- null.

Number

- Number is a primitive wrapper object.
- The JavaScript Number type is a double-precision 64-bit binary format IEEE 754 value
- . A Number only keeps about 17 decimal places of precision; arithmetic is subject to rounding
- The largest value a Number can hold is about 1.8×10308.
- Number may also be expressed in literal forms like 0b101, 0o13, 0×0A.

Numeric literals

- Decimal:
  - Number(333)⇒333;
  - But if you start a number with 0 .then it's take as a octal number. Number(0777)⇒511(take 0777 as a octal number)
- Exponential:
  - The decimal exponential literal is specified by the following format: beN; where b is a base number (integer or floating), followed by e char (which serves as separator or exponent indicator) and N, which is exponent or power number – a signed integer
  - Example:

```
0e-5   // => 0
0e+5   // => 0
Number(23e3)⇒23000
Number(43e-1)⇒4.3
```

- Binary :
  - Binary number syntax uses a leading zero followed by a lowercase or uppercase Latin letter "B" (0b or 0B).

  Example:

```
var FLT_SIGNBIT  = 0b10000000000000000000000000000000; // 2147483648
var FLT_EXPONENT = 0b01111111100000000000000000000000; // 2139095040
var FLT_MANTISSA = 0B00000000011111111111111111111111; // 8388607
```

- Octal:
  - Octal number syntax uses a leading zero followed by a lowercase or uppercase Latin letter "O" (0o or 0O).

  Example:

```
var n = 0O755; // 493
var m = 0o644; // 420

// Also possible with just a leading zero (see note about decimals above)
0755
0644
```

- Hexadecimal:
  - Hexadecimal number syntax uses a leading zero followed by a lowercase or uppercase Latin letter "X" (0x or 0X).
  - Example:

```
Number(0x5555)
21845
Number(0X2324)
8996
```

- BigInt:
- BigInt literals are created by appending n to the end of an integer.

```
Number(12123123141241234213412340)
1.2123123141241233e+24
Number(0x212312314124123421341234n)
1.0255403638971947e+28
```

Some extra about Number:

//max value : (2^53 -1)

//min value: (-(2^53 - 1)).

Static Methods:

1. Number.isNaN():

return true or false or undefined. Checks that the input is Not a Number or Number. NaN ⇒ Not a Number.

```
//the true valus of This process:
Number.isNaN(NaN);        // true
Number.isNaN(Number.NaN); // true
Number.isNaN(0 / 0);      // true

//false
Number.isNaN(true);
Number.isNaN(null);
Number.isNaN(37);
Number.isNaN('37');
Number.isNaN('37.37');
Number.isNaN('');
Number.isNaN(' ');
Number.isNaN('NaN');      // false
Number.isNaN(undefined);  // false
Number.isNaN({});         // false
Number.isNaN('blabla');   // false

//undefined:
console.log(typeOfNaN(Number('0x100F')
```

Polyfill:

```
Number.isNaN = Number.isNaN || function isNaN(input) {
    return typeof input === 'number' && input !== input;
}
```

2.Number.parseFloat(input):

The parseFloat() function parses an argument (converting it to a string first if needed) and returns a floating point number.

```
parseFloat({ toString: function() { return "3.14" } }); //3.14
parseFloat('3.14some non-digit characters');//(3.14)

parseFloat('432134214343chara213424234324234cters');
//432134214343
parseFloat('3.14');
//3.14
parseFloat('314e-2');
//3.14
parseFloat(3.14);//3.14


parseFloat('FF2');
NaN
```

3.Number.parseInt():

//it has basic 2 parts: 1. string and the other one is 2. radix.

it has it's own unique approach

```
parseInt(string)
parseInt(string, radix)
//radix  must be between 2 to 36
//if the string startswith 0 and dont provide any other information . then radix doesnot w
ork.it's basically work for octal notation

//radix is work for as a base.
```

Instance Methods:

1.Number.prototype.toExponential(fractionDigits):

it gives how many fractional number we put after floating point number.

```
function expo(x, f) {
  return Number.parseFloat(x).toExponential(f);
}
console.log(expo(123456, 2));
// expected output: "1.23e+5"//
console.log(expo(123456, 6));// "1.234560e+5"
console.log(expo('123456'));
// expected output: "1.23456e+5"

console.log(expo('oink'));
// expected output: "NaN"
```

2.Number.prototype.toLocaleString()

```
function eArabic(x){
  return x.toLocaleString('bn-EG');
}

console.log(eArabic(123456.789));
// expected output: "১,২৩,৪৫৬.৭৮৯"

console.log(eArabic('123456.789'));
// expected output: "123456.789"

console.log(eArabic(NaN));
// expected output: "ليس رقم"
```

3.Number.prototype.toFixed()

toFixed(digit)→ gives how many number are there after floating point

toFixed()→works as a parseInt();

```
function financial(x) {
  return Number.parseFloat(x).toFixed(2);
}

console.log(financial(123.456));
// expected output: "123.46"

console.log(Number.parseFloat(123.34321421341234).toFixed());
//123

console.log(financial(0.004));
// expected output: "0.00"

console.log(financial('1.23e+5'));
// expected output: "123000.00"
```

Math Operator(build a secondary calculator)

1.Math.floor()

It Returns the largest integer less than or equal to x.

```
console.log(Math.floor(123.12123));//123
console.log(Math.floor(123.9667876));//123
```

2.Math.abs()

Returns the absolute value of x.

```
console.log(Math.abs(-1312312))//1312312
```

3.Math.ceil()

Returns the smallest integer greater than or equal to x.

```
console.log(Math.ceil(123.9667876));//124
```

4.Math.sqrt()

Returns the positive square root of x.

```
console.log(Math.sqrt(25));//5
```

5.Math.pow(x,y)

Returns base x to the exponent power y (that is, x^y).

```
console.log(Math.pow(5,2));//25
```

6.Math.random()

Returns a pseudo-random number between 0 and 1.

```
console.log(Math.random());//0.696969696969
```

7.Math.round()

Returns the value of the number x rounded to the nearest integer.

```
console.log(Math.round(1.3423423));//1
console.log(Math.round(1.5423423));//2
```

8.Math.min([x,[,y,....]]])

Returns the smallest of zero or more numbers.

```
console.log(Math.max(8,914,12341234,12,321434,23434,342134))//8
```

9.Math.max([x[, y[, ...]]])

Returns the largest of zero or more numbers.

```
console.log(Math.max(8,914,12341234,12,321434,23434,342134))//12341234
```

10.Math.cbrt()

Returns the cube root of x.

```
console.log(Math.cbrt(64));//4
```

More at:

> Math - JavaScript | MDN
> Math is a built-in object that has properties and methods for mathematical constants and functions. It's not a function object. Unlike many other global objects, Math is not a constructor. All properties and methods of Math are static. You refer to the constant pi as Math.PI and you call the sine function as ,
> https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math

String

- Strings in JavaScript are sequences of Unicode characters
- they are sequences of UTF-16 code units; each code unit is represented by a 16-bit number.
- Each Unicode character is represented by either 1 or 2 code units.
- String can be represented as 2 variation:
  - as String: let s_prim = 'foo'
  - as Object: let s_obj = new String(s_prim)

```
let s_prim = 'foo'
let s_obj = new String(s_prim)

console.log(typeof s_prim) // Logs "string"
console.log(typeof s_obj)  // Logs "object"

//do some math rn
let s1 = '2 + 2'              // creates a string primitive
let s2 = new String('2 + 2')  // creates a String object
console.log(eval(s1))         // returns the number 4
console.log(eval(s2))         // returns the string "2 + 2"
```

Escape Sequence List

Escape sequence	Unicode code point
`\0`	null character (U+0000 NULL)
`\'`	single quote (U+0027 APOSTROPHE)
`\"`	double quote (U+0022 QUOTATION MARK)
`\\`	backslash (U+005C REVERSE SOLIDUS)
`\n`	newline (U+000A LINE FEED; LF)
`\r`	carriage return (U+000D CARRIAGE RETURN; CR)
`\v`	vertical tab (U+000B LINE TABULATION)
`\t`	tab (U+0009 CHARACTER TABULATION)
`\b`	backspace (U+0008 BACKSPACE)
`\f`	form feed (U+000C FORM FEED)

Static and Instance methods of string:

1. String.prototype.charAt(index):

   -Returns the character (exactly one UTF-16 code unit) at the specified index.

   -If index is out of range, charAt() returns an empty string

   -If no index is provided to charAt(), the default is 0.return th first element

```
const sentence = 'Kais Loves kais';

const index = 3;

console.log(`The character at index ${index} is ${sentence.charAt(index)}`);

//"The character at index 3 is s"
```

2. String.raw():

   The static String.raw() method is a tag function of template literals. This is similar to the r prefix in Python, or the @ prefix in C# for string literals

```
// Create a variable that uses a Windows
// path without escaping the backslashes:
const filePath = String.raw`C:\Development\profile\aboutme.html`;

console.log(`The file was uploaded from: ${filePath}`);
//"The file was uploaded from: C:\Development\profile\aboutme.html"
```

3. String.prototype.concat()

   The concat() method concatenates the string arguments to the calling string and returns a new string.

```
const str1 = 'Mark';
const str2 = 'Kais';

console.log(str1.concat(' ', str2));
// expected output: "Mark Kais"

console.log(str2.concat(', ', str1));
// expected output: "Kais, Mark"
```

4. String.prototype.includes()

   The includes() method performs a case-sensitive search to determine whether one string may be found within another string, returning true or false as appropriate.

```
includes(searchString)
includes(searchString, position)
searchString
  A string to be searched for within str.
position (Optional)
    The position within the string at which to begin searching for searchString. (Def
aults to 0.)
```

```
const sentence = 'The quick brown fox jumps over the lazy dog.';

const word = 'fox';

console.log(`The word "${word}" ${sentence.includes(word) ? 'is' : 'is not'} in the s
entence`);
// expected output: "The word "fox" is in the sentence"
```

5.String.prototype.endsWith()

The endsWith() method determines whether a string ends with the characters of a specified string, returning true or false as appropriate.

```
endsWith(searchString)
endsWith(searchString, length)
searchString
  The characters to be searched for at the end of str.
length Optional
    If provided, it is used as the length of str. Defaults to str.length.
```

```
const str1 = 'Cats are the best!';

console.log(str1.endsWith('best', 17));
// expected output: true

const str2 = 'Is this a question';

console.log(str2.endsWith('?'));
// expected output: false
let str = 'To be, or not to be, that is the question.'

console.log(str.endsWith('question.'))  // true
console.log(str.endsWith('to be'))      // false
console.log(str.endsWith('to be', 19))  // true
```

6.String.prototype.indexOf()

The indexOf() method returns the index within the calling String object of the first occurrence of the specified value, starting the search at fromIndex. Returns -1 if the value is not found.

```
indexOf(searchValue)
indexOf(searchValue, fromIndex)
fromIndex (Optional)
    An integer representing the index at which to start the search. Defaults to 0.
```

```
const paragraph = 'The quick brown fox jumps over the lazy dog. If the dog barked, was
it really lazy?';

const searchTerm = 'dog';
const indexOfFirst = paragraph.indexOf(searchTerm);

console.log(`The index of the first "${searchTerm}" from the beginning is ${indexOfFir
st}`);
// expected output: "The index of the first "dog" from the beginning is 40"

console.log(`The index of the 2nd "${searchTerm}" is ${paragraph.indexOf(searchTerm,
 (indexOfFirst + 1))}`);
// expected output: "The index of the 2nd "dog" is 52"
```

7.String.prototype.lastIndexOf():

The lastIndexOf() method returns the index within the calling String object of the last occurrence of the specified value, searching backwards from fromIndex. Returns -1 if the value is not found.

```
const paragraph = 'The quick brown fox jumps over the lazy dog. If the dog barked, was it
 really lazy?';

const searchTerm = 'dog';

console.log(`The index of the first "${searchTerm}" from the end is ${paragraph.lastIndexO
f(searchTerm)}`);
// expected output: "The index of the first "dog" from the end is 52"
```