The values are written as **name:value** pairs (name and value separated by a colon).

It is a common practice to declare objects with the const keyword.

E.g.

const person = { firstName : “John”, lastName : “Doe”, age : 50 }

Accessing Object Properties

User can access object properties in two ways :

1. objectName.propertyName
2. objectName[”propertyName”]

Object Methods

Object can also have methods.

Methods are actions that can be performed on objects.
Methods are stored in properties as function definitions.
E.g.

```
const person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

The this Keyword

In a function definition, this refers to the “owner” of the function.

In other words, this.firstName means the firstName property of the **this object**.

Accessing Object Methods

Syntax:

objectName.methodName()

E.g.

name = person.fullName();

JS Events

HTML events are “**things**” that happen to HTML elements.

When JavaScript is used in HTML pages, JavaScript can “**react**” on these events.

An HTML event can be something the browser does, or something a user does.

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

Common HTML Events

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushed a keyboard key
onload	The browser has finished loading the page

JS Strings

JavaScript strings are used for storing and manipulating text.

A JavaScript string is zero or more characters written inside quotes.

To find the length of a string, use the built-in length property.

E.g.

let x = “I’m ok”;
let y = ”I am ok”;
let z = “He is ‘Johnny’”

Escape Character

Because string must be written within quotes, JavaScript will misunderstand this string.

E.g.

let text = "We are the so-called "Vikings" from the north.";

Code	Result	Description
\’	‘	Single quote
\”	“	Double quote
\\	\	Backslash

Strings Can be Objects

Normally, JavaScript strings are primitive values, created from literals:

let firstName = “John”;

But strings can also be defined as objects with the keyword new:

let firstName = new String(“john”);

Note: Don’t create strings as objects. It slows down execution speed. The new keyword can complicate the code and produce unexpected results.

JS String Methods

String methods help user to work with strings.

String Methods and Properties

Primitive values, like “John Doe”, cannot have properties or methods (because they are not objects).

But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

Extracting String Parts

There are 3 methods for extracting a part of a string:

- slice(start, end)
- substring(start, end)
- substr(start, length)

The slice() Method

slice() extracts a part of a string and returns the extracted part in a new string.

The method takes 2 parameters : the start position, and the end position (end not included).

This example slices out a portion of a string from position 7 to position 12.
E.g.

```
let str = "Apple, Banana, Kiwi";
let part = str.slice(7, 13);
```

Note : JavaScript counts positions from zero. First position is 0.

If a parameter is negative, the position is counted from the end of the string.

If you omit the second parameter, the method will slice out the rest of the string, or counting from the end.

The substring() Method

substring() is similar to slice().

The difference is that substring() cannot accept negative indexes.

E.g.

```
let str = "Apple, Banana, Kiwi";
let part = substring(7, 13);
```

If user omit the second parameter, substring() will slice out the rest of the string.

The substr() Method

substr() is similar to slice().

The difference is that the second parameter specifies the length of the extracted part.

E.g.

```
let str = "Apple, Banana, Kiwi";
let part = str.substr(7, 6);
```

```
let str = "Apple, Banana, Kiwi";
let part = str.substr(7);
```

Replacing String Content

The replace() method replaces a specified value with another value in a string:

E.g.

```
let text = "Please visit Microsoft!";
let newText = text.replace("Microsoft", "W3Schools");
```

Note: The replace() method does not change the string it is called on. It returns a new string.
By default, the replace() method replaces only the first match. The replace() method is case sensitive.

To replace case insensitive, use a regular expression with an ” /i ” flag (insensitive).

To replace all matches, use a regular expression with a /g flag (global match).

Converting to Upper and Lower Case

A string is converted to upper case with toUpperCase().

E.g.

```
let text1 = "Hello World!";
let text2 = text1.toUpperCase();
```

A string is converted to lower case with toLowerCase().

E.g.

```
let text1 = "Hello World!";
let text2 = text1.toLowerCase();
```

The concat() Method

concat() joins two or more strings.

E.g.

```
let text1 = "Hello";
let text2 = "World";
let text3 = text1.concat(" ", text2);
```

The concat() method can be used instead of the plus operator. These two lines do the same.

Note: All string methods return a new string. They don't modify the original string.

Formally said: Strings are immutable: Strings cannot be changed, only replaced.

String.trim()

The trim() method removes whitespace from both sides of a string.

E.g.

```
let text = "       Hello World!        ";
text.trim()    // Returns "Hello World!"
```

JavaScript String Padding

ECMAScript 2017 added two String methods: padStart and padEnd to support padding at the beginning and at the end of a string.

E.g.

```
let text = "5";
text.padStart(4,0)    // Returns 0005
```

```
let text = "5";
text.padEnd(4,0)     // Returns 5000
```

Extracting String Characters

There are 3 methods for extracting string characters:

- charAt (position)
- charCodeAt (position)
- Property access [ ]

The charAt() Method

The charAt() method returns the character at a specified index (position) in a string.

E.g.

```
let text = "HELLO WORLD";
text.charAt(0)          // Returns H
```

The charCodeAt() Method

The charCodeAt() method returns the Unicode of the character at a specified index in a string.

E.g.

```
let text = "HELLO WORLD";

text.charCodeAt(0)      // Returns 72
```

Property Access

ECMAScript 5 (2009) allows property access [ ] on strings.

E.g.

```
let text = "HELLO WORLD";
text[0]                 // returns H
```

Converting a String to an Array

A string can be converted to an array with the split() method.

E.g.

text.split(“,”);
text.split(“”);

JS String Search

JavaScript methods for searching strings.

- String.indexOf()
- String.lastIndexOf()
- String.startsWith()
- String.endsWith()

String.indexOf()

The indexOf() method returns the index of (the position of) the first occurrence of a specified text in a string.

E.g.

```
let str = "Please locate where 'locate' occurs!";
str.indexOf("locate");
```

String.lastIndexOf()

The lastIndexOf() method returns the index of the last occurrence of a specified text in a string.

E.g.

```
let str = "Please locate where 'locate' occurs!";
str.lastIndexOf("locate");
```

Note: Both indexOf() and lastIndexOf() return -1 if the text is not found.

String.search()

The search() method searches a string for a specified value and returns the position of the match.

E.g.

```
let str = "Please locate where 'locate' occurs!";
str.search("locate");
```

The two methods, indexOf() and search() , are equal ?

They accept the same arguments (parameters), and return the same value ?

The two methods are NOT equal. These are the differences:

- The search() method cannot take a second start position argument.
- The indexOf() method cannot take powerful search values (regular expressions).

String.match()

The match() method search a string for a match against a regular expression, and returns the matches, as an Array object.

E.g.

```
let text = "The rain in SPAIN stays mainly in the plain";
text.match(/ain/g);
```

String.includes()

The includes() method returns true if a string contains a specified value.

E.g.

```
let text = "Hello world, welcome to the universe.";
text.includes("world");  // Return ” true “
```

String.startsWith()

The startsWith() method returns true if a string begins with a specified value, otherwise false.

E.g.

Note: The startsWith() method is case sensitive.

```
let text = "Hello world, welcome to the universe.";

text.startsWith("world", 6)    // Returns true
```

String.endsWith()

The endsWith() method returns true if a string ends with a specified value, otherwise false.

E.g.

```
var text = "John Doe";
text.endsWith("Doe");
```

JS String Templates Literals

Synonyms :

- Template Literals
- Template Strings
- String Templates
- Back-Tics Syntax

Back-Tics Syntax

Template Literals use back-ticks (```) rather than the quotes (“”) to define a string.

let text = `Hello World!` ;

Quotes Inside Strings

With template literals, user can use both single and double quotes inside a string.

E.g.

```
let text = `He's often called "Johnny” `;
```

Multiline Strings

Template literals allows multiline strings.

E.g.

```
let text =
`The quick
brown fox
jumps over
the lazy dog`;
```

Interpolation

Template literals provide an easy way to interpolate variables and expressions into strings.

The method is called string interpolation.

The syntax is :

${ ... }

Variable Substitutions

Template literals allow variables in strings.
E.g.

```
let firstName = "John";
let lastName = "Doe";

let text = `Welcome ${firstName} ${lastName}!`;
```

JS Numbers

JavaScript has only one type of number. Numbers can be written with or without decimals.

Extra large or extra small numbers can be written with scientific (exponent) notation.
E.g.

```
let x = 3.14;    // A number with decimals
let y = 3;       // A number without decimals

let a = 123e5;    // 12300000
let b = 123e-5;   // 0.00123
```

JavaScript Numbers are Always 64-bit Floating Point

Unlike many other programming languages, JavaScript does not define different types of numbers, like integers, short, long, floating-point etc.

JavaScript numbers are always stored as double precision floating point number, flowing the international IEEE 754 standard.

This format stores numbers in 64 bits, where the number (the fraction) is stored in bits 0 to 51 , the exponent in bits 52 to 62, and the sign in bit 63.

Value	Exponent	Sign
52 bits (0 - 51)	11 bits (52 – 62)	1 bit (63)

Precision

Integers (numbers without a period or exponent notation) are accurate up to 15 digits.

E.g.

```
let x = 999999999999999;   // x will be 999999999999999
let y = 9999999999999999;  // y will be 10000000000000000
```

The maximum number of decimals is 17, but floating point arithmetic is not always 100% accurate.

Numeric Strings

JavaScript strings can have numeric content.

JavaScript will try to convert strings to numbers in all numeric operations.

NaN – Not a Number

NaN is a JavaScript reserved word indicating that a number is not a legal number.

Trying to do arithmetic with a non-numeric string will result in NaN (Not a Number).

User can use the global JavaScript function isNaN() to find out if a value is a not a number.
E.g.

```
let x = 100 / "Apple";  // x will be NaN (Not a Number)
```

```
isNaN(x);
```

Note : if user use NaN in a mathematical operation, the result will also be NaN.

NaN is number: typeof NaN returns number.

Infinity

Infinity (or –Infinity) is the value JavaScript will return if user calculate a number outside the largest possible number.

Division by 0 also generates Infinity.

Infinity is a number : typeof Infinity returns number.

Hexadecimal

JavaScript interprets numeric constant as hexadecimal if they are preceded by 0x.

E.g.

```
let x = 0xFF;       // x will be 255
```

By default, JavaScript displays numbers as base 10 decimals.

But user can use the toString() method to output numbers from base 2 to base 36.

Hexadecimal is base 16. Decimal is base 10. Octal is base 8. Binary is base2.
E.g.

```
let myNumber = 32;
myNumber.toString(10);  // returns 32
myNumber.toString(32);  // returns 10
myNumber.toString(16);  // returns 20
myNumber.toString(8);   // returns 40
myNumber.toString(2);   // returns 100000
```

JS Number Methods

The toString() Method

The toString() method returns a number as a string.

All number methods can be used on any type of number (literals, variables, or expressions)

The toExponential() Method

toExponential() returns a string, with a number rounded and written using exponential notation.

A parameter defines the number of characters behind the decimal point.
E.g.

```
let x = 9.656;
x.toExponential(2);     // returns 9.66e+0
```

The toFixed() Method

toFixed() returns a string, with the number written with a specified number of decimals.

E.g.

```
let x = 9.656;
x.toFixed(0);          // returns 10
x.toFixed(2);          // returns 9.66
x.toFixed(4);          // returns 9.6560
x.toFixed(6);          // returns 9.656000
```

The toPrecision() Method

toPrecision() returns a string, with a number written with a specified length.

E.g.

```
let x = 9.656;
x.toPrecision();       // returns 9.656
x.toPrecision(2);      // returns 9.7
x.toPrecision(4);      // returns 9.656
```

The valueOf() Method

valueOf() returns a number as a number.

E.g.

```
let x = 123;
x.valueOf();            // returns 123 from variable x
(123).valueOf();        // returns 123 from literal 123
(100 + 23).valueOf();   // returns 123 from
```

Note: All JavaScript data types have a valueOf() and a toString() method.

Converting Variables to Numbers

There are 3 JavaScript methods that can be used to convert variables to numbers:

- The Number() method
- The parseInt() method
- The parseFloat() method

These methods are not number methods, but global JavaScript methods.

E.g.

```
Number(true);        // returns 1
Number(false);       // returns 0
Number("10");        // returns 10
Number("  10");      // returns 10
Number("10  ");      // returns 10
Number(" 10  ");     // returns 10
Number("10.33");     // returns 10.33
```

Note: If the number cannot be converted, NaN (Not a Number) is returned.

E.g.

```
parseInt("-10");       // returns -10
parseInt("-10.33");    // returns -10
parseInt("10");        // returns 10
parseInt("10.33");     // returns 10
parseInt("10 20 30");  // returns 10
parseInt("10 years");  // returns 10
parseInt("years 10");  // returns NaN
```

E.g.

```
parseFloat("10");       // returns 10
parseFloat("10.33");    // returns 10.33
parseFloat("10 20 30");  // returns 10
parseFloat("10 years");  // returns 10
parseFloat("years 10");  // returns NaN
```