Numbers

* JavaScript supports both integer and floating-point numbers that can be represented in decimal, hexadecimal or octal notation.
* Unlike other languages, JavaScript does not treat integer and floating-point numbers differently.
* All numbers in JavaScript are represented as floating-point numbers.
* Numbers can also be represented in hexadecimal notation (base 16).
* Hexadecimal numbers are prefixed with 0x.

*Parsing Integers from Strings

* parseInt() Can be used to parse an integer from a string.
* This method is particularly handy in situations when you are dealing with the values like CSS units e.g. 50px, 12pt, etc. and you would like to extract the numeric value out of it.
* If the parseInt() method encounters a character that is not numeric in the specified base, it stops parsing and returns the integer value parsed up to that point.
* If the first character cannot be converted into a number, the method will return NaN (not a number). Leading and trailing spaces are allowed.

*Numbers

* Converting Numbers to Strings
* toString() can be used to convert a number to its string equivalent.
* This method optionally accepts an integer parameter in the range 2 through 36 specifying the base to use for representing numeric values.

*Numbers

* Formatting Numbers to Fixed Decimals
* toFixed() can be used to format a number with a fixed number of digits to the right of the decimal point.
* The value returned by this method is a string and it has exactly specified number of digits after the decimal point.
* If the digits parameter is not specified or omitted, it is treated as 0.

*The Number Object

*The Number object represents numerical date, either integers or floating-point numbers

*Math.round() Math.round(x) returns the value of x rounded to its nearest integer

*Math.pow() Math.pow(x, y) returns the value of x to the power of y:

*Math.sqrt() Math.sqrt(x) returns the square root of x:

*Math.abs(): Ensures the number is always positive

* Math.random() returns a random number between 0 (inclusive), and 1 (exclusive)

* Math.random()

*The Boolean Object

*The Boolean object represents two values, either "true" or "false". If value parameter is omitted or is 0, null, false, NaN, undefined, or the empty string (" "), the object has an initial value of false.

*Strings

*The String object lets you work with a series of characters; it wraps JavaScript's string primitive data type with a number of helper methods.

*charAt Return the character at the specified position in string.

*indexOf, lastIndexOf Return the position of specified substring in the string or last position of specified substring, respectively.

*startsWith, endsWith, includes Returns whether or not the string starts, ends or contains a specified string.

*Concat Combines the text of two strings and returns a new string.

*slice Extracts a section of a string and returns a new string.

* Substring Return the specified subset of the string, either by specifying the start and end indexes or the start index and a length.
* toLowerCase, toUpperCase Return the string in all lowercase or all uppercase, respectively.
* Repeat Returns a string consisting of the elements of the object repeated the given times.
* trim Trims whitespace from the beginning and end of the string.

* Strings

* Template literals are enclosed by the back-tick (` `) (grave accent) character instead of double or single quotes. Template literals can contain place holders. These are indicated by the Dollar sign and curly braces (${expression}).

* Multi-line template literals

*Date

*The Date object is a datatype built into the JavaScript language. Date objects are created with the new Date ().

*Once a Date object is created, a number of methods allow you to operate on it. Most methods simply allow you to get and set the year, month, day, hour, minute, second, and millisecond fields of the object, using either local time or UTC (universal, or GMT) time.

*Date Methods

*Date() Returns today's date and time

*getDate() Returns the day of the month for the specified date according to the local time

*getDay() Returns the day of the week for the specified date according to the local time

*getFullYear() Returns the year of the specified date according to the local time

*getHours() Returns the hour in the specified date according to the local time

*getMilliseconds() Returns the milliseconds in the specified date according to the local time

*getMinutes() Returns the minutes in the specified date according to the local time

*getMonth() Returns the month in the specified date according to the local time

*getSeconds() Returns the seconds in the specified date according to the local time

*Conditional statements

* Use the if statement to execute a statement if a logical condition is true. Use the optional else clause to execute a statement if the condition is false.
* may also compound the statements using else if to have multiple conditions tested in sequence

*Comparison operators

* Equal (==) Returns true if the operands are equal.
* Not equal (!=) Returns true if the operands are not equal.
* Strict equal (===) Returns true if the operands are equal and of the same type .
* Strict not equal (!==) Returns true if the operands are of the same type but not equal or different type.
* Greater than (>) Returns true if the left operand is greater than the right operand.
* Greater than or equal (>=) Returns true if the left operand is greater than or equal to the right operand.
* Less than (<) Returns true if the left operand is less than the right operand.
* Less than or equal (<=) Returns true if the left operand is less than or equal to the right operand.

* A switch statement allows a program to evaluate an expression and attempt to match the expression's value to a case label. If a match is found, the program executes the associated statement

* switch statement

Loops

* At times, certain instructions require repeated execution. Loops are an ideal way to do the same. A loop represents a set of instructions that must be repeated. In a loop's context, a repetition is termed as an iteration.

*for statement

*A for loop repeats until a specified condition evaluates to false

```
for ([initialExpression]; [condition]; [incrementExpression])
statement
```

*The initializing expression initialExpression, if any, is executed. This expression usually initializes one or more loop counters, This expression can also declare variables.

*If the value of condition is true, the loop statements execute. If the value of condition is false, the for loop terminates.

*The statement executes. To execute multiple statements, use a block statement ({ ... }) to group those statements.

*If present, the update expression incrementExpression is executed.

*Control returns to step 2.

*do...while statement

*The do...while statement repeats until a specified condition evaluates to false

```
do statement while (condition);
```

*statement executes once before the condition is checked. To execute multiple statements, use a block statement ({ ... }) to group those statements. If condition is true, the statement executes again. At the end of every execution, the condition is checked. When the condition is false, execution stops and control passes to the statement following do...while.

while statement

* A while statement executes its statements as long as a specified condition evaluates to true.

```
while (condition) statement
```

* The condition test occurs before statement in the loop is executed. If the condition returns true, statement is executed and the condition is tested again. If the condition returns false, execution stops and control is passed to the statement following while.
* To execute multiple statements, use a block statement ({ ... }) to group those statements.

* Use the break statement to terminate a loop, switch
* The continue statement can be used to restart a while, do-while, for statement.

* Break-continue statement