Javascript Basics

## **Comments**

### Single Line Comments //

Single line comments start with //

// Im a single line comments

### Multi-line Comments /\* \*/

Multi-line comments start with /\* and end with \*/

/\*

Im a Multi line comment

\*/

## **JavaScript Identifiers / Names**

Identifiers are JavaScript names.

Identifiers are used to name variables and keywords, and functions.

A JavaScript name must begin with:

* A letter (A-Z or a-z)
* A dollar sign ($)
* Or an underscore (\_)

Subsequent characters may only contain letters, digits, underscores, or dollar signs.

Best practice to use camelcase for creating an identifier. thisIsCamelCase

Best practice for creating a constant identifier to name it in all caps. PI = 3.14

## **Variables**

There are 3 Ways to Declare a JavaScript Variable:

var, let, const, nothing

const text = “hello”;

The const keyword should be used by default as it is reduces bugs since its variable cannot be changed..

The var keyword is used in all JavaScript code from 1995 to 2015.

The let and const keywords were added to JavaScript in 2015.

Const is for constants and let is for variables that will change.

## **JavaScript Data Types**

There are eight basic data types in JavaScript. They are String, Number, BigInt, Boolean, undefined, null, symbol and object:

| Data Types | Description | Example |
| --- | --- | --- |
| String | represents textual data | 'hello', "hello world!" etc |
| Number | an integer or a floating-point number | 3, 3.234, 3e-2 etc. |
| BigInt | an integer with arbitrary precision | 900719925124740999n , 1n etc. |
| Boolean | Any of two values: true or false | true and false |
| undefined | a data type whose variable is not initialized | let a; |
| null | denotes a null/empty value | let a = null; |
| Symbol | data type whose instances are unique and immutable | let value = Symbol('hello'); |
| Object | key-value pairs of collection of data | let student = { }; |

Typeof

The typeof operator returns a string indicating the data type of a value

typeof 3

>>>”Number”

### JavaScript strings

JavaScript strings are for storing and manipulating text.

In JavaScript, strings are immutable. That means the characters of a string cannot be changed. However, you can assign the variable name to a new string.

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

* Single quotes: 'Hello'
* Double quotes: "Hello"
* Backticks: `Hello`

String() takes null and undefined and converts them to string. However, toString() gives error when null are passed.

#### Convert to string

To convert other data types to strings, you can use either String() or toString()

String() takes null and undefined and converts them to string. However, toString() gives error when null are passed.

#### JavaScript String Methods

All string methods return a new value. They do not change the original variable.

| **Name** | **Description** |
| --- | --- |
| [charAt()](https://www.w3schools.com/jsref/jsref_charat.asp) | Returns the character at a specified index (position) |
| [charCodeAt()](https://www.w3schools.com/jsref/jsref_charcodeat.asp) | Returns the Unicode of the character at a specified index |
| [concat()](https://www.w3schools.com/jsref/jsref_concat_string.asp) | Returns two or more joined strings |
| [constructor](https://www.w3schools.com/jsref/jsref_constructor_string.asp) | Returns the string's constructor function |
| [endsWith()](https://www.w3schools.com/jsref/jsref_endswith.asp) | Returns if a string ends with a specified value |
| [fromCharCode()](https://www.w3schools.com/jsref/jsref_fromcharcode.asp) | Returns Unicode values as characters |
| [includes()](https://www.w3schools.com/jsref/jsref_includes.asp) | Returns if a string contains a specified value |
| [indexOf()](https://www.w3schools.com/jsref/jsref_indexof.asp) | Returns the index (position) of the first occurrence of a value in a string |
| [lastIndexOf()](https://www.w3schools.com/jsref/jsref_lastindexof.asp) | Returns the index (position) of the last occurrence of a value in a string |
| [length](https://www.w3schools.com/jsref/jsref_length_string.asp) | Returns the length of a string |
| [localeCompare()](https://www.w3schools.com/jsref/jsref_localecompare.asp) | Compares two strings in the current locale |
| [match()](https://www.w3schools.com/jsref/jsref_match.asp) | Searches a string for a value, or a regular expression, and returns the matches |
| [prototype](https://www.w3schools.com/jsref/jsref_prototype_string.asp) | Allows you to add properties and methods to an object |
| [repeat()](https://www.w3schools.com/jsref/jsref_repeat.asp) | Returns a new string with a number of copies of a string |
| [replace()](https://www.w3schools.com/jsref/jsref_replace.asp) | Searches a string for a value, or a regular expression, and returns a string where the values are replaced |
| [search()](https://www.w3schools.com/jsref/jsref_search.asp) | Searches a string for a value, or regular expression, and returns the index (position) of the match |
| [slice()](https://www.w3schools.com/jsref/jsref_slice_string.asp) | Extracts a part of a string and returns a new string |
| [split()](https://www.w3schools.com/jsref/jsref_split.asp) | Splits a string into an array of substrings |
| [startsWith()](https://www.w3schools.com/jsref/jsref_startswith.asp) | Checks whether a string begins with specified characters |
| [substr()](https://www.w3schools.com/jsref/jsref_substr.asp) | Extracts a number of characters from a string, from a start index (position) |
| [substring()](https://www.w3schools.com/jsref/jsref_substring.asp) | Extracts characters from a string, between two specified indices (positions) |
| [toLocaleLowerCase()](https://www.w3schools.com/jsref/jsref_tolocalelowercase.asp) | Returns a string converted to lowercase letters, using the host's locale |
| [toLocaleUpperCase()](https://www.w3schools.com/jsref/jsref_tolocaleuppercase.asp) | Returns a string converted to uppercase letters, using the host's locale |
| [toLowerCase()](https://www.w3schools.com/jsref/jsref_tolowercase.asp) | Returns a string converted to lowercase letters |
| [toString()](https://www.w3schools.com/jsref/jsref_tostring_string.asp) | Returns a string or a string object as a string |
| [toUpperCase()](https://www.w3schools.com/jsref/jsref_touppercase.asp) | Returns a string converted to uppercase letters |
| [trim()](https://www.w3schools.com/jsref/jsref_trim_string.asp) | Returns a string with removed whitespaces |
| [trimEnd()](https://www.w3schools.com/jsref/jsref_string_trim_end.asp) | Returns a string with removed whitespaces from the end |
| [trimStart()](https://www.w3schools.com/jsref/jsref_string_trim_start.asp) | Returns a string with removed whitespaces from the start |
| [valueOf()](https://www.w3schools.com/jsref/jsref_valueof_string.asp) | Returns the primitive value of a string or a string object |

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### JavaScript Number

Number represents integer and floating numbers (decimals and exponentials). For example, 1, 3.6

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

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

#### JavaScript Number() Function

The Number() function is used to convert various data types to numbers

#### JavaScript Number Methods and Properties

All number methods return a new value. They do not change the original value.

| **Name** | **Description** |
| --- | --- |
| [constructor](https://www.w3schools.com/jsref/jsref_constructor_number.asp) | Returns the function that created JavaScript's Number prototype |
| [EPSILON](https://www.w3schools.com/jsref/jsref_number_epsilon.asp) | Returns the difference between 1 and the smallest number greater than 1 |
| [isFinite()](https://www.w3schools.com/jsref/jsref_isfinite_number.asp) | Checks whether a value is a finite number |
| [isInteger()](https://www.w3schools.com/jsref/jsref_isinteger.asp) | Checks whether a value is an integer |
| [isNaN()](https://www.w3schools.com/jsref/jsref_isnan_number.asp) | Checks whether a value is Number.NaN |
| [isSafeInteger()](https://www.w3schools.com/jsref/jsref_issafeinteger.asp) | Checks whether a value is a safe integer |
| [MAX\_SAFE\_INTEGER](https://www.w3schools.com/jsref/jsref_max_safe_integer.asp) | Returns the maximum safe integer in JavaScript. |
| [MIN\_SAFE\_INTEGER](https://www.w3schools.com/jsref/jsref_min_safe_integer.asp) | Returns the minimum safe integer in JavaScript |
| [MAX\_VALUE](https://www.w3schools.com/jsref/jsref_max_value.asp) | Returns the largest number possible in JavaScript |
| [MIN\_VALUE](https://www.w3schools.com/jsref/jsref_min_value.asp) | Returns the smallest number possible in JavaScript |
| [NaN](https://www.w3schools.com/jsref/jsref_number_nan.asp) | Represents a "Not-a-Number" value |
| [NEGATIVE\_INFINITY](https://www.w3schools.com/jsref/jsref_negative_infinity.asp) | Represents negative infinity (returned on overflow) |
| [POSITIVE\_INFINITY](https://www.w3schools.com/jsref/jsref_positive_infinity.asp) | Represents infinity (returned on overflow) |
| [parseFloat()](https://www.w3schools.com/jsref/jsref_number_parsefloat.asp) | Parses a string an returns a number |
| [parseInt()](https://www.w3schools.com/jsref/jsref_number_parseint.asp) | Parses a string an returns a whole number |
| [prototype](https://www.w3schools.com/jsref/jsref_prototype_num.asp) | Allows you to add properties and methods to an object |
| [toExponential(x)](https://www.w3schools.com/jsref/jsref_toexponential.asp) | Converts a number into an exponential notation |
| [toFixed(x)](https://www.w3schools.com/jsref/jsref_tofixed.asp) | Formats a number with x numbers of digits after the decimal point |
| [toLocaleString()](https://www.w3schools.com/jsref/jsref_tolocalestring_number.asp) | Converts a number into a string, based on the locale settings |
| [toPrecision(x)](https://www.w3schools.com/jsref/jsref_toprecision.asp) | Formats a number to x length |
| [toString()](https://www.w3schools.com/jsref/jsref_tostring_number.asp) | Converts a number to a string |
| [valueOf()](https://www.w3schools.com/jsref/jsref_valueof_number.asp) | Returns the primitive value of a number |

### 

### 

### 

### 

### 

### 

### 

### 

### JavaScript Booleans

JavaScript booleans can have one of two values: true or false.

#### The Boolean() Function

You can use the Boolean() function to find out if an expression (or a variable) is true or change a data type to a boolean data type.

Everything With a "Value" will evaluate as true whereas Everything Without a "Value" (0, NULL, Undefined, “”, NaN, false) will evaluate false

## 

## **JavaScript Operators**

### JavaScript Comparison Operators

A comparison operator compares its operands and returns a logical boolean value (true or false) based on whether the comparison is true. The operands can be numerical, string, logical, or object values.

| **Operator** | **Description** |
| --- | --- |
| == | equal to |
| === | equal value and equal type (use === instead of ==) |
| != | not equal |
| !== | not equal value or not equal type (use !== instead of !=) |
| > | greater than |
| < | less than |
| >= | greater than or equal to |
| <= | less than or equal to |
| ? | ternary operator |

### 

### JavaScript Arithmetic Operators

**Arithmetic Operators** are used to perform arithmetic on numbers

| **Operator** | **Description** |
| --- | --- |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| \*\* | Exponentiation ([ES2016](https://www.w3schools.com/js/js_2016.asp)) |
| / | Division |
| % | Modulus (Division Remainder) |
| ++ | Increment |
| -- | Decrement |

### JavaScript Logical Operators

| **Operator** | **Description** |
| --- | --- |
| && | logical and |
| || | logical or |
| ! | logical not |

### 

### Operator precedence

**Operator precedence** determines how operators are parsed concerning each other. Operators with higher precedence become the operands of operators with lower precedence.

| **Precedence** | **Operator type** | **Associativity** | **Individual operators** |
| --- | --- | --- | --- |
| 18 | [Grouping](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Grouping) | n/a | ( … ) |
| 17 | [Member Access](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Property_accessors#dot_notation) | left-to-right | … . … |
| [Optional chaining](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Optional_chaining) | … ?. … |
| [Computed Member Access](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Property_accessors#bracket_notation) | n/a | … [ … ] |
| [new](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new) (with argument list) | new … ( … ) |
| [Function Call](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions) | … ( … ) |
| 16 | [new](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new) (without argument list) | n/a | new … |
| 15 | [Postfix Increment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#increment_and_decrement) | n/a | … ++ |
| [Postfix Decrement](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#increment_and_decrement) | … -- |
| 14 | [Logical NOT (!)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Logical_NOT) | n/a | ! … |
| [Bitwise NOT (~)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Bitwise_NOT) | ~ … |
| [Unary plus (+)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Unary_plus) | + … |
| [Unary negation (-)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Unary_negation) | - … |
| [Prefix Increment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#increment_and_decrement) | ++ … |
| [Prefix Decrement](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#increment_and_decrement) | -- … |
| [typeof](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/typeof) | typeof … |
| [void](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/void) | void … |
| [delete](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/delete) | delete … |
| [await](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/await) | await … |
| 13 | [Exponentiation (\*\*)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Exponentiation) | right-to-left | … \*\* … |
| 12 | [Multiplication (\*)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Multiplication) | left-to-right | … \* … |
| [Division (/)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Division) | … / … |
| [Remainder (%)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Remainder) | … % … |
| 11 | [Addition (+)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Addition) | left-to-right | … + … |
| [Subtraction (-)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Subtraction) | … - … |
| 10 | [Bitwise Left Shift (<<)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Left_shift) | left-to-right | … << … |
| [Bitwise Right Shift (>>)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Right_shift) | … >> … |
| [Bitwise Unsigned Right Shift (>>>)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Unsigned_right_shift) | … >>> … |
| 9 | [Less Than (<)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Less_than) | left-to-right | … < … |
| [Less Than Or Equal (<=)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Less_than_or_equal) | … <= … |
| [Greater Than (>)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Greater_than) | … > … |
| [Greater Than Or Equal (>=)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Greater_than_or_equal) | … >= … |
| [in](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/in) | … in … |
| [instanceof](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/instanceof) | … instanceof … |
| 8 | [Equality (==)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Equality) | left-to-right | … == … |
| [Inequality (!=)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Inequality) | … != … |
| [Strict Equality (===)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Strict_equality) | … === … |
| [Strict Inequality (!==)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Strict_inequality) | … !== … |
| 7 | [Bitwise AND (&)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Bitwise_AND) | left-to-right | … & … |
| 6 | [Bitwise XOR (^)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Bitwise_XOR) | left-to-right | … ^ … |
| 5 | [Bitwise OR (|)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Bitwise_OR) | left-to-right | … | … |
| 4 | [Logical AND (&&)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Logical_AND) | left-to-right | … && … |
| 3 | [Logical OR (||)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Logical_OR) | left-to-right | … || … |
| [Nullish coalescing operator (??)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Nullish_coalescing) | … ?? … |
| 2 | [Assignment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators#assignment_operators) | right-to-left | … = … |
| … += … |
| … -= … |
| … \*\*= … |
| … \*= … |
| … /= … |
| … %= … |
| … <<= … |
| … >>= … |
| … >>>= … |
| … &= … |
| … ^= … |
| … |= … |
| … &&= … |
| … ||= … |
| … ??= … |
| [Conditional (ternary) operator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Conditional_operator) | right-to-left  (Groups on expressions after ?) | … ? … : … |
| [Arrow (=>)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions) | right-to-left | … => … |
| [yield](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/yield) | n/a | yield … |
| [yield\*](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/yield*) | yield\* … |
| [Spread (...)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax) | ... … |
| 1 | [Comma / Sequence](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comma_operator) | left-to-right | … , … |

## **Conditional Statements**

you can use the JavaScript if...else statement to create a program that can make decisions.

In JavaScript we have the following conditional statements:

* Use if to specify a block of code to be executed, if a specified condition is true
* Use else to specify a block of code to be executed, if the same condition is false
* Use else if to specify a new condition to test, if the first condition is false
* Use switch to specify many alternative blocks of code to be executed

### if Statement

The syntax of the if statement is:

if (condition) {

// the body of if

}

The if statement evaluates the condition inside the parenthesis ().

1. If the condition is evaluated to true, the code inside the body of if is executed.
2. If the condition is evaluated to false, the code inside the body of if is skipped.

### if...else statement

An if statement can have an optional else clause. The syntax of the if...else statement is:

if (condition) {

// block of code if condition is true

} else {

// block of code if condition is false

}

### The else if Statement

Use the else if statement to specify a new condition if the first condition is false.

if (*condition1*) {

// *block of code to be executed if condition1 is true*

} else if (*condition2*) {

// *block of code to be executed if the condition1 is false and condition2 is true*

} else {

// *block of code to be executed if the condition1 is false and condition2 is false*

}

### switch statement

The JavaScript switch statement is used in decision making.

The switch statement evaluates an expression and executes the corresponding body that matches the expression's result.

The syntax of the switch statement is:

switch(variable/expression) {

case value1:

// body of case 1

break;

case value2:

// body of case 2

break;

case valueN:

// body of case N

break;

default:

// body of default

}

The switch statement evaluates a variable/expression inside parentheses ().

* If the result of the expression is equal to value1, its body is executed.
* If the result of the expression is equal to value2, its body is executed.
* This process goes on. If there is no matching case, the default body executes.

### 

### Ternary operator

A ternary operator evaluates a condition and executes a block of code based on the condition.

Its syntax is:

condition ? expression1 : expression2

The ternary operator evaluates the test condition.

* If the condition is true, **expression1** is executed.
* If the condition is false, **expression2** is executed.

# 

# 

# 

# 

## 

## 

## **JavaScript Iterations (Loops)**

Loops can execute a block of code a number of times.

JavaScript supports different kinds of loops:

* for - loops through a block of code a number of times
* for/in - loops through the properties of an object
* for/of - loops through the values of an iterable object
* while - loops through a block of code while a specified condition is true
* do/while - also loops through a block of code while a specified condition is true

### **JavaScript for loop**

The syntax of the for loop is:

for (initialExpression; condition; updateExpression) {

// for loop body

}

1. The **initialExpression** initializes and/or declares variables and executes only once.
2. The **condition** is evaluated.
   * If the condition is false, the for loop is terminated.
   * If the condition is true, the block of code inside of the for loop is executed.
3. The **updateExpression** updates the value of **initialExpression** when the condition is true.
4. The **condition** is evaluated again. This process continues until the condition is false.

Example:

for (let rep = 1; rep <= 30; rep++) {

console.log(`Lifting weights repetition ${rep} 🏋️‍♀️`);

}

## **JavaScript while Loop**

The syntax of the while loop is:

while (condition) {

// body of loop

}

1. A while loop evaluates the **condition** inside the parenthesis ().
2. If the **condition** evaluates to true, the code inside the while loop is executed.
3. The **condition** is evaluated again.
4. This process continues until the **condition** is false.
5. When the **condition** evaluates to false, the loop stops.

Example

let dice = Math.trunc(Math.random() \* 6) + 1;

while (dice !== 6) {

console.log(`You rolled a ${dice}`);

dice = Math.trunc(Math.random() \* 6) + 1;

if (dice === 6) console.log('Loop is about to end...');

}

# 

# 

# 

# 

## 

## **JavaScript Functions**

### **Declarative Functions**

A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses **()**.

function function*Name*() {

// *code to be executed*

}

The parentheses may include parameter names separated by commas:

**(*parameter1, parameter2, ...*)**

**function function*Name*(*parameter1, parameter2, parameter3*) {**

**// *code to be executed***

**}**

Example:

function add(a, b) {

console.log(a + b);

}

// calling functions

add(3,4);

add(2,9);

In the above program, the add function is used to find the sum of two numbers.

* The function is declared with two parameters a and b.
* The function is called using its name and passing two arguments **3** and **4** in one and **2** and **9** in another.
* Inside the function, the arguments (the parameters) behave as local variables.

### **Function Return**

The return statement can be used to return the value to a function call.

The return statement denotes that the function has ended. Any code after return is not executed.

If nothing is returned, the function returns an undefined value.

function add(a, b) {

return a + b;

}

### **Function Expressions**

In Javascript, functions can also be defined as expressions. For example,

// program to find the square of a number

// function is declared inside the variable

let x = function (num) { return num \* num };

console.log(x(4));

// can be used as variable value for other variables

let y = x(3);

console.log(y);

In the above program, variable *x* is used to store the function. Here the function is treated as an expression. And the function is called using the variable name.

The function above is called an anonymous function.

**Note:** In ES2015, JavaScript expressions are written as arrow functions.

### **Arrow Functions**

Arrow function is one of the features introduced in the ES6 version of JavaScript. It allows you to create functions in a cleaner and shorter way compared to regular functions.

The syntax of the arrow function is:

let myFunction = () => {

statement(s)

}

If the body has only one statement, you can write arrow function as:

let myFunction = (parameter1, parameter2) => expression

The this keyword is not associated with arrow functions unlike declarative and expressive functions.

## **Javascript Data Structures**

### **JavaScript Arrays**

An array is a special variable, which can hold more than one value:

const cars = ["Saab", "Volvo", "BMW"];

#### **Accessing the First Array Element**

### **Example**

const fruits = ["Banana", "Orange", "Apple", "Mango"];

let fruit = fruits[0];

#### **Accessing the Last Array Element**

### **Example**

const fruits = ["Banana", "Orange", "Apple", "Mango"];

let fruit = fruits[fruits.length - 1];

#### **How to Recognize an Array**

A common question is: How do I know if a variable is an array?

The problem is that the JavaScript operator typeof returns "object":

To solve this problem ECMAScript 5 (JavaScript 2009) defined a new method Array.isArray():

Array.isArray(fruits);

#### **JavaScript Array Methods and Properties**

| **Name** | **Description** |
| --- | --- |
| [at()](https://www.w3schools.com/jsref/jsref_array_at.asp) | Returns an indexed element of an array |
| [concat()](https://www.w3schools.com/jsref/jsref_concat_array.asp) | Joins arrays and returns an array with the joined arrays |
| [constructor](https://www.w3schools.com/jsref/jsref_constructor_array.asp) | Returns the function that created the Array object's prototype |
| [copyWithin()](https://www.w3schools.com/jsref/jsref_copywithin.asp) | Copies array elements within the array, to and from specified positions |
| [entries()](https://www.w3schools.com/jsref/jsref_entries.asp) | Returns a key/value pair Array Iteration Object |
| [every()](https://www.w3schools.com/jsref/jsref_every.asp) | Checks if every element in an array pass a test |
| [fill()](https://www.w3schools.com/jsref/jsref_fill.asp) | Fill the elements in an array with a static value |
| [filter()](https://www.w3schools.com/jsref/jsref_filter.asp) | Creates a new array with every element in an array that pass a test |
| [find()](https://www.w3schools.com/jsref/jsref_find.asp) | Returns the value of the first element in an array that pass a test |
| [findIndex()](https://www.w3schools.com/jsref/jsref_findindex.asp) | Returns the index of the first element in an array that pass a test |
| [flat()](https://www.w3schools.com/jsref/jsref_array_flat.asp) | Concatenates sub-array elements |
| [flatMap()](https://www.w3schools.com/jsref/jsref_array_flatmap.asp) | Maps all array elements and creates a new flat array |
| [forEach()](https://www.w3schools.com/jsref/jsref_foreach.asp) | Calls a function for each array element |
| [from()](https://www.w3schools.com/jsref/jsref_from.asp) | Creates an array from an object |
| [includes()](https://www.w3schools.com/jsref/jsref_includes_array.asp) | Check if an array contains the specified element |
| [indexOf()](https://www.w3schools.com/jsref/jsref_indexof_array.asp) | Search the array for an element and returns its position |
| [isArray()](https://www.w3schools.com/jsref/jsref_isarray.asp) | Checks whether an object is an array |
| [join()](https://www.w3schools.com/jsref/jsref_join.asp) | Joins all elements of an array into a string |
| [keys()](https://www.w3schools.com/jsref/jsref_keys.asp) | Returns a Array Iteration Object, containing the keys of the original array |
| [lastIndexOf()](https://www.w3schools.com/jsref/jsref_lastindexof_array.asp) | Search the array for an element, starting at the end, and returns its position |
| [length](https://www.w3schools.com/jsref/jsref_length_array.asp) | Sets or returns the number of elements in an array |
| [map()](https://www.w3schools.com/jsref/jsref_map.asp) | Creates a new array with the result of calling a function for each array element |
| [pop()](https://www.w3schools.com/jsref/jsref_pop.asp) | Removes the last element of an array, and returns that element |
| [prototype](https://www.w3schools.com/jsref/jsref_prototype_array.asp) | Allows you to add properties and methods to an Array object |
| [**push()**](https://www.w3schools.com/jsref/jsref_push.asp) | Adds new elements to the end of an array, and returns the new length |
| [reduce()](https://www.w3schools.com/jsref/jsref_reduce.asp) | Reduce the values of an array to a single value (going left-to-right) |
| [reduceRight()](https://www.w3schools.com/jsref/jsref_reduceright.asp) | Reduce the values of an array to a single value (going right-to-left) |
| [reverse()](https://www.w3schools.com/jsref/jsref_reverse.asp) | Reverses the order of the elements in an array |
| [shift()](https://www.w3schools.com/jsref/jsref_shift.asp) | Removes the first element of an array, and returns that element |
| [slice()](https://www.w3schools.com/jsref/jsref_slice_array.asp) | Selects a part of an array, and returns the new array |
| [some()](https://www.w3schools.com/jsref/jsref_some.asp) | Checks if any of the elements in an array pass a test |
| [sort()](https://www.w3schools.com/jsref/jsref_sort.asp) | Sorts the elements of an array |
| [splice()](https://www.w3schools.com/jsref/jsref_splice.asp) | Adds/Removes elements from an array |
| [toString()](https://www.w3schools.com/jsref/jsref_tostring_array.asp) | Converts an array to a string, and returns the result |
| [unshift()](https://www.w3schools.com/jsref/jsref_unshift.asp) | Adds new elements to the beginning of an array, and returns the new length |
| [valueOf()](https://www.w3schools.com/jsref/jsref_valueof_array.asp) | Returns the primitive value of an array |

# 

## **JavaScript Objects**

Objects are variables too. But objects can contain many values.

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

The syntax to declare an object is:

const object\_name = {

key1: value1,

key2: value2

}

Here, an object object\_name is defined. Each member of an object is a **key: value** pair separated by commas and enclosed in curly braces {}

In JavaScript, "key: value" pairs are called **properties**. For example,

let person = {

name: 'John',

age: 20

};

Here, name: 'John' and age: 20 are properties.

### **Accessing Object Properties**

You can access the **value** of a property by using its **key**. This can be done using dot notation or bracket notation.

#### **1. Using dot Notation**

Here's the syntax of the dot notation. objectName.key

For example,

const person = {

name: 'John',

age: 20,

};

console.log(person.name); // John

#### **2. Using bracket Notation**

Here is the syntax of the bracket notation. objectName["propertyName"]

For example,

const person = {

name: 'John',

age: 20,

};

console.log(person["name"]);

### **Object Methods (Functions)**

In JavaScript, objects can also contain functions. For example,

const person = {

name: 'John',

greet: function() { console.log('hello'); }

};

In the above example, a person object has two keys (name and greet), which have a string value and a function value, respectively.

### **Accessing Object Methods**

You can access an object method using a dot notation. The syntax is:

objectName.methodKey()

You can access property by calling an **objectName** and a **key**. You can access a method by calling an **objectName** and a **key** for that method along with ()

person.greet(); // hello

Here, the greet method is accessed as person.greet()

### **JavaScript this Keyword**

To access a property of an object from within a method of the same object, you need to use the this keyword.

In order to access the **properties** of an object, this keyword is used following by . and **key**.

const mark = {

fullName: "Mark Miller",

mass: 78,

height: 1.69,

calcBMI : function (){

this.bmi = **this.**mass / (**this.**height \* **this.**height);

return this.bmi;

}

};

mark.calcBMI();

console.log(mark.bmi);

>>>27.3

## **Strict mode**

“use strict”;

Strict mode is declared by adding "use strict"; to the beginning of a script or a function (strict mode has to be the first thing in the program. The only thing that can go before it is comments.)

Declared at the beginning of a script, it has global scope (all code in the script will execute in strict mode):

You can use strict mode in all your programs. It helps you to write clean and secure code, like preventing you from using undeclared variables.

JavaScript's strict mode is a way to *opt in* to a restricted variant of JavaScript, thereby implicitly opting-out of "[sloppy mode](https://developer.mozilla.org/en-US/docs/Glossary/Sloppy_mode)". Strict mode isn't just a subset: it *intentionally* has different semantics from normal code. Browsers not supporting strict mode will run strict mode code with different behavior from browsers that do, so don't rely on strict mode without feature-testing for support for the relevant aspects of strict mode.