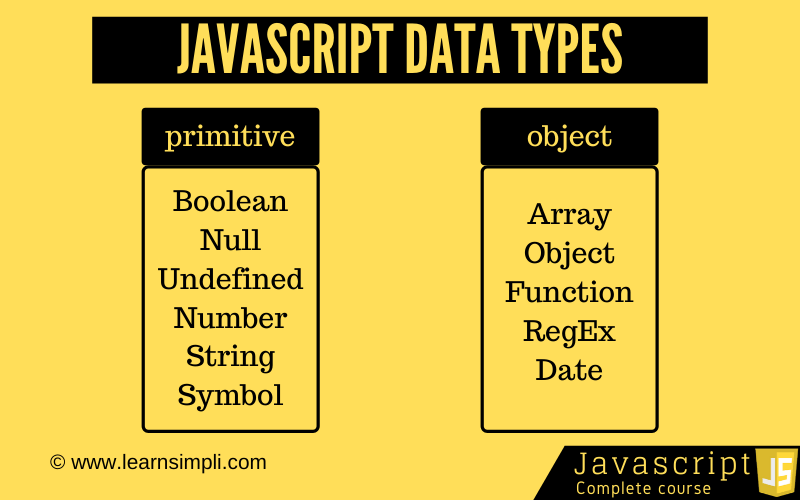
|  |  |  |  |
| --- | --- | --- | --- |
| **What** | **Why- How** | **Code Example** | |
| Var | Variables declared with var are in the function scope.  Hoisting  is allowed in var - means that you can define a variable before its declaration.  Reassigning is allowed in var -To reassign a value is to reassign the value of a variable.  Redeclaration is allowed in var - The redeclaration of a variable means that you can declare the variable again. | var v1 = 1;  v1 = 21; //reassigning the var | |
| let | Variables declared as let are in the block scope.  Reassigning is allowed in let - To reassign a value is to reassign the value of a variable. | let v1 = 1;  v1 = 21; //reassigning the let | |
| const | Variables declared as const are in the block scope. | Const v1 = 1; | |
| Array- | An array is a special variable, which can hold more than one value: | let array = [ 1, "one", 2, "two", 3, "three"]; | |
| Splice Method | The splice() method adds new items to an array. | array.splice(3, 4, "TWO", "THREE"); | |
| Slice Method | The slice() method slices out a piece o an array. | let newarray = array.slice(3); | |
| Push Array | It adds given value in the end | array.push(4, "four") | |
| Shift Array | it removed the first element from an array | array.shift() | |
| Unshift Array | adds new element in an array at first place | array.unshift(5, "five"); | |
| Array length | to get array length | (array.length); | |
| Array pop |  | (array.pop()); | |
| New Array |  | (newarray); | |
| Empty Array |  | let emptyarray = [];  emptyarray[0] = 6, "six";  emptyarray[1] = 7, "seven"; | |
| Concat method | merge 2 array and create  a new array | let concatArray = array.concat(emptyarray); | |
| Array Sorting | sorts array alphabetically/numerically | sortarray.sort(); | |
| Array Reverse | The reverse() method reverses the elements in an array | sortarray.reverse(); | |
| This Keyword | This keyword refers to an object. | var employee =  {      Name: "Hardik",      DOJ: 10-12-2022,      employeedetails: function() {          return this.Name + this.DOJ;     }  };  console.log(employee.employeedetails()); | |
| Strict Mode | In a function, in strict mode, this is undefined. Strict mode makes it easier to write "secure" JavaScript. Strict mode changes previously accepted "bad syntax" into real errors.  As an example, in normal JavaScript, mistyping a variable name creates a new global variable. In strict mode, this will throw an error, making it impossible to accidentally create a global variable.  In normal JavaScript, a developer will not receive any error feedback assigning values to non-writable properties.  In strict mode, any assignment to a non-writable property, a getter-only property, a non-existing property, a non-existing variable, or a non-existing object, will throw an error. | "use strict"; myFunction();  function myFunction() {   y = 3.14;   // This will also cause an error because y is not declared } | |
| AJAX | AJAX is a developer's dream, because you can:  Update a web page without reloading the page  Request data from a server - after the page has loaded  Receive data from a server - after the page has loaded  Send data to a server - in the background |  | |
| Jquery $ sign | Jquery $ sign is just a valid JavaScript identifier which is used as an alias for jQuery. | (function( $ ){  $.fn.myfunction = function() {  alert('hello world');  return this;  };  })( jQuery ); | |
| Https request in ajax | The XMLHttpRequest object is used to exchange data with a server.To send a request to a server, we use the open() and send() methods of the XMLHttpRequest object: |  | |
| Status in AJAX HTML | HTTP response status codes indicate whether a specific HTTP request has been successfully completed. |  | |
| Understanding of Asycnhronous behaviour | With asynchronous programming, the user can move to another screen while the function continues to execute. When a photo is loaded and sent on Instagram, the user does not have to stay on the same screen waiting for the photo to finish loading. | Callbacks, Promises, and Async/Await | |
| For loop | Loops are handy, if you want to run the same code over and over again, each time with a different value.  Often this is the case when working with arrays: | for (expression 1; expression 2; expression 3) {   // code block to be executed }  // program to display numbers from 1 to 5  const n = 5;  // looping from i = 1 to 5  // in each iteration, i is increased by 1  for (let i = 1; i <= n; i++) {  console.log(i); // printing the value of i  } | |
| if else | The if-else or conditional statement will perform some action for a specific condition. If the condition meets then a particular block of action will be executed otherwise it will execute another block of action that satisfies that particular condition. Such control statements are used to cause the flow of execution to advance and branch based on changes to the state of a program.  JavaScript’s conditional statements:  if  if-else  nested-if  if-else-if ladder | var i = 10;        if (i < 15)       console.log("i is less than 15");      else       console.log("I am Not in if");  if (condition1)  {  // Executes when condition1 is true  if (condition2)  {  // Executes when condition2 is true  }  } | |
| math.random | Math.random() returns a random number between 0 (inclusive),  and 1 (exclusive): | // Returns a random number: Math.random(); | |
| Regular expressions | Regular expressions are patterns used to match character combinations in strings. In JavaScript, regular expressions are also objects. These patterns are used with the | These patterns are used with the exec() and test() methods of RegExp, and with the match(), matchAll(), replace(), replaceAll(), search(), and split() methods of String. | |
| pure and impure functions |  |  | |
| constructor  Function | In a constructor function this does not have a value. It is a substitute for the new object. The value of this will become the new object when a new object is created. |  | |
| Eventloop | JavaScript has a runtime model based on an event loop, which is responsible for executing the code, collecting and processing events, and executing queued sub-tasks. This model is quite different from models in other languages like C and Java. |  | |
| Arrow function | It gets shorter! If the function has only one statement, and the statement returns a value, you can remove the brackets and the return keyword: | let myFunction = (a, b) => a \* b; | |
| window.onload | This function execute a JavaScript immediately after a page has been loaded: |  | |
| Fetching API |  | let p = fetch("https: api.chucknorris.io/jokes/random%22)  p.then((response) => {    return response.json()  }).then((value2) => {    console.log(value2)    const jokeData = value2.value;    console.log(jokeData);  }) | |
| async and await | async and await make promises easier to write" async makes a function return a Promise await makes a function wait for a Promise | const getData = async() => {    var y = await "Hello World";    console.log(y);  }  console.log(1);  getData();  console.log(2); | |
| Promises | - Promises are used to handle asynchronous operations in JavaScript. They are easy to manage when dealing with multiple asynchronous operations where callbacks can create callback hell leading to unmanageable code.  A Promise has four states:  fulfilled: Action related to the promise succeeded  rejected: Action related to the promise failed  pending: Promise is still pending i.e. not fulfilled or rejected yet  settled: Promise has fulfilled or rejected  Benefits of Promises  Improves Code Readability  Better handling of asynchronous operations  Better flow of control definition in asynchronous logic  Better Error Handling | var promise = new Promise(function(resolve, reject){        do something  }); | |
| iterators and generators |  |  | |
| differnce between map and for each |  |  | |
| Dynamic typing: | One thing you might have noticed that we are using the var or let for creating the strings, numbers and null. Its because javascript is a loosely typed language. Let’s see the below examples | let employeeName = 'John';  console.log(employeeName);  employeeName = 'Mickel';  console.log(employeeName);  employeeName = 'Stark';  console.log(employeeName); | |
| Operators | Operators perform some kind of operation on the left and right side operands. | | |
| Arithmetic operator: | The arithmetic operator can perform the addition, subtraction, multiplication and division. | | // addition //subtraction //multiplication//division  let left = 2;  let right = 2;  let result = left + right; OR  let result = left - right; OR  let result = left \* right; OR  let result = left / right;  console.log(result); |
| Logical operators: | Logical operators are AND (&&) and OR (||). These operators can be used to evaluate one or more conditions to perform some actions. | | // logical  let user = {  name: 'John',  isAdmin: true,  isActive : true  };  if(user.isAdmin === true && user.isActive === true) OR if(user.isAdmin === true || user.isActive === true)  {  console.log('User has admin access');  } else {  console.log('User dont have admin access');  } |
| Typeof operator in Javascript: | Typeof is a kind of operator in Javascript that return type of the variable. Let’s look at some examples | | let number = 1;  let string = 'Name';  let isAdmin = true;  let user = {  name: 'John'  };  console.log(typeof number);  console.log(typeof string);  console.log(typeof isAdmin);  console.log(typeof user);  // out put  // number  // string  // boolean |
| Ternary operator: | The ternary operator is shorthand of if-else condition. The logic is the same as the if-else condition, but it can be done in a single line. Let’s look at some examples. It is also called as a conditional operator. | | let user = {  name: 'John',  isAdmin: true,  isActive : false  };  let doesUserHasAccess = user.isAdmin ? 'User has admin access' : 'User doesnt have admin access';  console.log(doesUserHasAccess);  // output  // User has admin access |

**Data Types in JS**

Primitive Values

There are 7 different data types that are primitives in the latest ECMAScript standard and they are All types except [Object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#objects) define [immutable](https://developer.mozilla.org/en-US/docs/Glossary/Immutable) values represented directly at the lowest level of the language. We refer to values of these types as primitive values.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type** | **What it does.** | **typeof return value** | **Object wrapper** | **Code Example** |
| [Null](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#null_type) | Null: A data type variable that notifies variable points to none | "object" | N/A | var employeeName = null; |
| [Undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#undefined_type) | Undefined: A data type variable that does not have a value yet | "undefined" | N/A | var employeeName; |
| [Boolean](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#boolean_type) | Boolean: A data type represents true or false | "boolean" | [Boolean](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Boolean) | const isAdmin = new Boolean(true); |
| [Number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#number_type) | Number: A data type represents floating-point numbers, for decimals and integers | "number" | [Number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number) | const x = 1; |
| [BigInt](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#bigint_type) | BigInt: A data type represents large integers even beyond the safe integer limit for Numbers | "bigint" | [BigInt](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/BigInt) |  |
| [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#string_type) | String: A data type represents a sequence of characters used for text. | "string" | [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String) | const employeeName = 'Mickel'; |
| [Symbol](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#symbol_type) | Symbol: A Symbol is a unique and immutable primitive value | "symbol" | [Symbol](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Symbol) |  |

Non Primitives values:  
All objects such as functions, arrays and structured JSON.

**Functions In jS**

|  |  |  |
| --- | --- | --- |
| **What** | **Why- How** | **Code Example** |
| A function in Javascript: | A piece of code to execute small task or operations  The function is an instance of an object type  So the Function almost behaves like other objects  Any Function can be assigned to variables  We can pass one function to another function as an argument  We can return a function from a function  The function can be defined with the keyword function  Then followed by function name with parenthesis  Arguments can be passed in parenthesis  Then followed by open and closed curly braces.  We can pass 0 or n number of arguments | function fullName(firstName, lastName) {  return firstName + ' '+ lastName;  }  console.log(fullName('John','Mickel'));  // output  // John Mickel |
|  | Now we will at look the point “we can pass one function to another function as an argument” and write an example for the same | function netPayable(salary, callBack) {  var netPay = [];  for (let i=0;i< salary.length;i++){  netPay.push(callBack(salary[i]));  }  return netPay;  }  function getPayable(salary){  return salary - 200;  }  var salaries = [30000,40000,50000,24000];  var payableSalaries = netPayable(salaries,getPayable);  console.log(payableSalaries);  // output  // (4) [29800, 39800, 49800, 23800] |
|  | Now look at the point “we can return a function from a function” and write an example for the same | function getExamDate(student){  if(student === 'be'){  return function(name){  console.log('Dear '+name + ' your exams will start on monday 10:30 AM...');  }  } else if (student === 'medical'){  return function(name){  console.log('Dear '+name + ' your exams will start on tuesday 10:30 AM...');  }  } else {  console.log('Dear '+name + ' your exams will start on thursday 10:30 AM...');  }  }  var beStudent = getExamDate('be');  beStudent('Stark');  var medicalStudent = getExamDate('medical');  medicalStudent('Mickel');  // output  // Dear Stark your exams will start on monday 10:30 AM...  // Dear Mickel your exams will start on tuesday 10:30 AM... |

Objects In jS

|  |  |  |
| --- | --- | --- |
| What | Why- How | Code Example |
| Objects | JavaScript has an object entity with a combination of properties.  An object has the key-value pair.  Every element has the name which is called key  Objects help to group together different variable that belongs to the same person or entity.  In object, the order doesn’t matter  The object in javascript can be created using the keyword [Object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects) | //Employee object  var Employee = new Object();  Employee.name = 'Name';  Employee.designation = 'CEO';  Employee.email = 'name@gmail.com'; |
| Constructor | With the help of constructor function: You can create an object with the new keyword with constructor | function Person(name, age, sex) {  this.name = name;  this.age = age;  this.sex = sex;  }  var Person1 = new Person('Rajani', 33, 'M');  console.log(Person1.name);  // output  // Rajani |
|  |  |  |
|  |  |  |