Javascript

The prototype object is special type of enumerable object to which additional properties can be attached to it which will be shared across all the instances of it's constructor function.
eCMA International publishes the specification for scripting languages is called 'ECMAScript'. ECMAScript specification defined in ECMA-262 for creating a general-purpose scripting language.
Use the online editor to quickly execute the JavaScript code without any installation. The followings are free online editors:
jsfiddle.net
jsbin.com
playcode.io
Case Sensitive
JavaScript is a case-sensitive scripting language. So, name of functions, variables and keywords are case sensitive. For example, myfunction and MyFunction are different, Name is not equal to nAme, etc.
Variables
In JavaScript, a variable is declared with or without the var keyword
JavaScript ignores multiple spaces and tabs
Write comment after double slashes // or write multiple lines of comments between /* and */
JavaScript Message Boxes: alert(), confirm(), prompt()
Sometimes you may need to take the user's input to do further actions.
The alert(), confirm(), and prompt() functions are global functions. So, they can be called using the window object e.g. window.alert(), window.confirm(), and window.prompt().
JavaScript is a loosely typed language. It means it does not require a data type to be declared. You can assign any literal values to a variable, e.g., string, integer, float, boolean, etc.
The variables declared without the var keyword becomes global variables, irrespective of where they are declared. Visit Variable Scope in JavaScript to learn about it.
It is Not Recommended to declare a variable without var keyword because it can accidentally overwrite an existing global variable.

Points to Remember
Variable stores data value that can be changed later on.
Variables can be defined using var keyword. Variables defined without var keyword become global variables.
Variables must be initialized before accessing it.
JavaScript allows multiple white spaces and line breaks in a variable declaration.
Multiple variables can be defined in a single line separated by a comma.
JavaScript is a loosely-typed language, so a variable can store any type value.
Variable names are case-sensitive.
Variable names can contain letters, digits, or the symbols $ and _. It cannot start with a digit 0 - 9.
Variables can have local or global scope. Local variables cannot be accessed out of the function where they are declared, whereas the global variables can be accessed from anywhere.
Points to Remember :
JavaScript object is a standalone entity that holds multiple values in terms of properties and methods.
Object property stores a literal value and method represents function.
An object can be created using object literal or object constructor syntax.
Object literal:

```
var person = {
   firstName: "James",
   lastName: "Bond",
   age: 25,
 getFullName: function () {
       return this.firstName + ' ' + this.lastName
       }
};
```

Object constructor:

```
var person = new Object();

person.firstName = "James";
person["lastName"] = "Bond";
```

```
person.age = 25;
person.getFullName = function () {
        return this.firstName + ' ' + this.lastName;
    };
```

Object properties and methods can be accessed using dot notation or [ ] bracket.
An object is passed by reference from one function to another.
An object can include another object as a property.
The ++ and -- operators are unary operators.
===	Compares equality of two operands with type.
JavaScript provides a special operator called ternary operator :? that assigns a value to a variable based on some condition
Points to Remember:
JavaScript includes operators that perform some operation on single or multiple operands (data value) and produce a result.
JavaScript includes various categories of operators: Arithmetic operators, Comparison operators, Logical operators, Assignment operators, Conditional operators.
Ternary operator ?: is a short form of if-else condition.
JavaScript is dynamic and loosely typed language. It means you don't require to specify a type of a variable. A variable in JavaScript can be assigned any type of value, as shown in the following example.

```
var myvariable = 1;  // numeric value
myvariable = 'one'; // string value
myvariable = 1.1; // decimal value
myvariable = true; // Boolean value
myvariable = null; // null valuePrimitive Data Types
```

The primitive data types are the lowest level of the data value in JavaScript. The typeof operator can be used with primitive data types to know the type of a value.

Data Type	Description
String	String is a textual content wrapped inside ' ' or " " or ` ` (tick sign). Example: 'Hello World!', "This is a string", etc.
Number	Number is a numeric value.

Example: 100, 4521983, etc.
BigInt BigInt is a numeric value in the arbitrary precision format.
Example: 453889879865131n, 200n, etc.
Boolean	Boolean is a logical data type that has only two values, true or false.
Null	A null value denotes an absense of value.
Example: var str = null;
Undefined	undefined is the default value of a variable that has not been assigned any value.
Example: In the variable declaration, var str;, there is no value assigned to str. So, the type of str can be check using typeof(str) which will return undefined.
Structural Data Types
The structural data types contain some kind of structure with primitive data.
Data Type	Description
Object	An object holds multiple values in terms of properties and methods.
Example:

```
var person = {
            firstName: "James",
            lastName: "Bond",
            age: 15
        };
```

Date	Date object represents date & time including days, months, years, hours, minutes, seconds and milliseconds.
Example: var today = new Date("25 July 2021");
Array	An array stores multiple values using special syntax.
Example: var nums = [1, 2, 3, 4];
Learn about each data type in detail in the next section.
If you want to include same quotes in a string value as surrounding quotes then use backward slash (\) before quotation mark inside string value.
Points to Remember:
JavaScript string must be enclosed in double or single quotes (" " or ' ').
String can be assigned to a variable using = operator.
Multiple strings can be concatenated using + operator.
A string can be treated as character array.

Use back slash (\) to include quotation marks inside string.
String objects can be created using new keyword. e.g. var str = new String();
String methods are used to perform different task on strings.
Points to Remember :
An array is a special type of variable that stores multiple values using a special syntax.
An array can be created using array literal or Array constructor syntax.
Array literal syntax: var stringArray = ["one", "two", "three"];
Array constructor syntax:var numericArray = new Array(3);
A single array can store values of different data types.
An array elements (values) can be accessed using zero based index (key). e.g. array[0].
An array index must be numeric.
Array includes length property and various methods to operate on array objects.The following table lists all the Array methods.

Method	Description
concat()	Returns new array by combining values of an array that is specified as parameter with existing array values.
every()	Returns true or false if every element in the specified array satisfies a condition specified in the callback function. Returns false even if single element does not satisfy the condition.
filter()	Returns a new array with all the elements that satisfy a condition specified in the callback function.
forEach()	Executes a callback function for each elements of an array.
indexOf()	Returns the index of the first occurrence of the specified element in the array, or -1 if it is not found.
join()	Returns string of all the elements separated by the specified separator
lastIndexOf()	Returns the index of the last occurrence of the specified element in the array, or -1 if it is not found.
map()	Creates a new array with the results of calling a provided function on every element in this array.
pop()	Removes the last element from an array and returns that element.

push()	Adds one or more elements at the end of an array and returns the new length of the array.
reduce()	Pass two elements simultaneously in the callback function (till it reaches the last element) and returns a single value.
reduceRight()	Pass two elements simultaneously in the callback function from right-to-left (till it reaches the last element) and returns a single value.
reverse()	Reverses the elements of an array. Element at last index will be first and element at 0 index will be last.
shift()	Removes the first element from an array and returns that element.
slice()	Returns a new array with specified start to end elements.
some()	Returns true if at least one element in this array satisfies the condition in the callback function.
sort()	Sorts the elements of an array.
splice()	Adds and/or removes elements from an array.
toString()	Returns a string representing the array and its elements.
unshift()	Adds one or more elements to the front of an array and returns the new length of the array.

Undefined is also a primitive value in JavaScript. A variable or an object has an undefined value when no value is assigned before using it. So you can say that undefined means lack of value or unknown value.

Points to Remember :

null and undefined are primitive values in JavaScript.

A null value means absence.

An undefined value means lack of value.

A null or undefined value evalutes to false in conditional expression.

Anonymous function is useful in passing callback function, creating closure or Immediately invoked function expression.

```
var showMessage = function (){
   alert("Hello World!");
};
```

Points to Remember :

JavaScript a function allows you to define a block of code, give it a name and then execute it as many times as you want.

A function can be defined using function keyword and can be executed using () operator.
A function can include one or more parameters. It is optional to specify function parameter values while executing it.
JavaScript is a loosely-typed language. A function parameter can hold value of any data type.
You can specify less or more arguments while calling function.
All the functions can access arguments object by default instead of parameter names.
A function can return a literal value or another function.
A function can be assigned to a variable with different name.
JavaScript allows you to create anonymous functions that must be assigned to a variable.
points to Remember :
Use if-else conditional statements to control the program flow.
JavaScript includes three forms of if condition: if condition, if else condition and else if condition.
The if condition must have conditional expression in brackets () followed by single statement or code block wrapped with { }.
'else if' statement must be placed after if condition. It can be used multiple times.
'else' condition must be placed only once at the end. It must come after if or else if statement.
Points to Remember :
The switch is a conditional statement like if statement.
A switch statement includes literal value or is expression based
A switch statement includes multiple cases that include code blocks to execute.
A break keyword is used to stop the execution of case block.
A switch case can be combined to execute same code block for multiple cases.

```
for (; ;) {
  if (i >= 5)
    break;
    console.log(arr[i]);
   i++;
```

}

Points to Remember :

JavaScript while loop & do-while loop execute the code block repeatedly till conditional expression returns true.

do-while loop executes the code at least once even if condition returns false.

Points to Remember :

JavaScript has global scope and local scope.

Variables declared and initialized outside any function become global variables.

Variables declared and initialized inside function becomes local variables to that function.

Variables declared without var keyword inside any function becomes global variables automatically.

Global variables can be accessed and modified anywhere in the program.

Local variables cannot be accessed outside the function declaration.

Global variable and local variable can have same name without affecting each other.

JavaScript does not allow block level scope inside { } brackets

eval() is a global function in JavaScript that evaluates a specified string as JavaScript code and executes it.

```
var str = '({"firstName":"Bill","lastName":"Gates"})';
var obj = eval(str);
obj.firstName; // Bill
```

It is not recommended to use eval() because it is slow, not secure, and makes code unreadable and maintainable.

JavaScript is a loosely-typed language. It does not give compile-time errors. So some times you will get a runtime error for accessing an undefined variable or calling undefined function etc.

```
try
{
   // code that may throw an error
}
catch(ex)
{
   // code to be executed if an error occurs
}
finally{
   // code to be executed regardless of an error occurs or not
}
```

throw

Use throw keyword to raise a custom error.

```
try
{
   throw "Error occurred";
}
catch(ex)
{
   alert(ex);
}
"use strict";
var x = 1; // valid in strict mode
y = 1; // invalid in strict mode
```

The strict mode in JavaScript does not allow following things:

Use of undefined variables

Use of reserved keywords as variable or function name

Duplicate properties of an object

Duplicate parameters of function

Assign values to read-only properties

Modifying arguments object

Octal numeric literals

with statement

eval function to create a variable

Strict mode can be applied to function level in order to implement strictness only in that particular function.

```
x = 1; //validfunction sum(val1, val2){
   "use strict";
    result = val1 + val2; //error
   return result;
}
```

JavaScript Hoisting

Hoisting is a concept in JavaScript, not a feature. In other scripting or server side languages, variables or functions must be declared before using it.

In JavaScript, variable and function names can be used before declaring it. The JavaScript compiler moves all the declarations of variables and functions at the top so that there will not be any error. This is called hoisting.

```
x = 1;
alert('x = ' + x); // display x = 1
var x;
```

Example: Hoisting not applicable for initialized variables

```
alert('x = ' + x); // display x = undefined
var x = 1;
```

Hoisting of Function

JavaScript compiler moves the function definition at the top in the same way as variable declaration.

Example: Function Hoisting

```
alert(Sum(5, 5)); // 10
function Sum(val1, val2)
{
   return val1 + val2;
}
Add(5, 5); // error
var Add = function Sum(val1, val2)
{
   return val1 + val2;
}
```

Points to Remember :

JavaScript compiler moves variables and function declaration to the top and this is called hoisting.

Only variable declarations move to the top, not the initialization.

Functions definition moves first before variables.

JavaScript ECMAScript 5, does not have class type. So it does not support full object oriented programming concept as other languages like Java or C#. However, you can create a function in such a way so that it will act as a class. the following example demonstrates how a function can be used like a class in JavaScript.
Example: Class in JavaScript

```
function Person() {
   this.firstName = "unknown";
   this.lastName = "unknown";
}
var person1 = new Person();
person1.firstName = "Steve";
person1.lastName = "Jobs";
alert(person1.firstName + " " + person1.lastName);var person2 = new Person();
person2.firstName = "Bill";
person2.lastName = "Gates";

alert(person2.firstName + " " + person2.lastName );Add Methods in a Class
```

We can add a function expression as a member variable in a function in JavaScript. This function expression will act like a method of class.Example: Method in Class

```
function Person() {
         this.firstName = "unknown";
         this.lastName = "unknown";
         this.getFullName = function(){
            return this.firstName + " " + this.lastName;
         }
      };var person1 = new Person();person1.firstName = "Steve";
person1.lastName = "Jobs";
alert(person1.getFullName());

var person2 = new Person();
person2.firstName = "Bill";
```

```
person2.lastName = "Gates";
alert(person2.getFullName());
```

In the above example, the Person function includes function expression that is assigned to a member variable getFullName. So now, getFullName() will act like a method of the Person class. It can be called using dot notation e.g. person1.getFullName().

Multiple Properties

Specify more than one property in defineProperties() method as shown below.

Example: Multiple Properties

```
function Person(firstName, lastName, age) {
   var _firstName = firstName || "unknown";
   var _lastName = lastName || "unknown";
   var _age = age || 25;
   Object.defineProperties(this, {
      "FirstName": {
         get: function () { return _firstName },
         set: function (value) { _firstName = value }
      },
      "LastName": {
         get: function () { return _lastName },
         set: function (value) { _lastName = value }
      },
      "Age": {
         get: function () { return _age },
         set: function (value) { _age = value }
      }
   });
   this.getFullName = function () {
         return this.FirstName + " " + this.LastName;
   }
};var person1 = new Person();
person1.FirstName = "John";
person1.LastName = "Bond";
```

alert(person1.getFullName());As you know, object in JavaScript can be created using object literal, object constructor or constructor function. Object includes properties. Each property can either be assigned a literal value or a function.//

```
object literal
var person = {
  firstName:'Steve',
  lastName:'Jobs'
};// Constructor function
function Student(){
  this.name = "John";
  this.gender = "Male";
  this.sayHi = function(){
    alert('Hi');
  }
}
var student1 = new Student();
console.log(student1.name);
console.log(student1.gender);
student1.sayHi();
```

https://www.tutorialsteacher.com/javascript/javascript-object-in-depthhttps://www.tutorialsteacher.com/javascript/this-keyword-in-javascripthttps://www.tutorialsteacher.com/javascript/new-keyword-in-javascriptSo, use prototype property of a function in the above example in order to have age properties across all the objects as shown below.Example: prototype

```
function Student() {
    this.name = 'John';
    this.gender = 'M';
}Student.prototype.age = 15;var studObj1 = new Student();
alert(studObj1.age); // 15var studObj2 = new Student();
alert(studObj2.age); // 15The magic of this is closure. In other words, the sayHi()
```

function is a closure.A closure is a function that preserves the outer scope in its inner scope.function greeting() {

```
    let message = 'Hi';    function sayHi() {
        console.log(message);
    }    return sayHi;
}
let hi = greeting();
hi(); //
```

Prototype in JavaScript

JavaScript is a dynamic language. You can attach new properties to an object at any time as shown below.Example: Attach property to object

```
function Student() {
    this.name = 'John';
    this.gender = 'Male';
}var studObj1 = new Student();
studObj1.age = 15;
alert(studObj1.age); // 15var studObj2 = new Student();
alert(studObj2.age); // undefined
```

As you can see in the above example, age property is attached to studObj1 instance. However, studObj2 instance will not have age property because it is defined only on studObj1 instance.So what to do if we want to add new properties at later stage to a function which will be shared across all the instances?The answer is Prototype.The prototype is an object that is associated with every functions and objects by default in JavaScript, where function's prototype property is accessible and modifiable and object's prototype property (aka attribute) is not visible.Every function includes prototype object by default.The prototype object is special type of enumerable object to which additional properties can be attached to it which will be shared across all the instances of it's constructor function.So, use prototype property of a function in the above example in order to have age properties across all the objects as shown below.Example: prototype

```
function Student() {
    this.name = 'John';
    this.gender = 'M';
}Student.prototype.age = 15;var studObj1 = new Student();
```

alert(studObj1.age); // 15var studObj2 = new Student();
alert(studObj2.age); // 15Both for..in and for..of are looping constructs which are used to iterate over data structures. The only difference between them is the entities they iterate over:for..in iterates over all enumerable property keys of an object
for..of iterates over the values of an iterable object. Examples of iterable objects are arrays, strings, and NodeLists.
JavaScript | Promises
Difficulty Level : Easy
Last Updated : 06 Dec, 2021
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. Prior to promises events and callback functions were used but they had limited functionalities and created unmanageable code.
Multiple callback functions would create callback hell that leads to unmanageable code. Also it is not easy for any user to handle multiple callbacks at the same time.
Benefits of Promises
Improves Code Readability
Better handling of asynchronous operations
Better flow of control definition in asynchronous logic
Better Error Handling
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
A promise can be created using Promise constructor.
Syntax

```
var promise = new Promise(function(resolve, reject){
    //do something
});A promise can be created using Promise constructor.
```

Syntax

```
var promise = new Promise(function(resolve, reject){
    //do something
});
```

Parameters

Promise constructor takes only one argument which is a callback function (and that callback function is also referred as anonymous function too).

Callback function takes two arguments, resolve and reject

Perform operations inside the callback function and if everything went well then call resolve.

If desired operations do not go well then call reject.

```
var promise = new Promise(function(resolve, reject) {
  const x = "geeksforgeeks";
  const y = "geeksforgeeks"
  if(x === y) {
    resolve();
  } else {
    reject();
  }
});

promise.
    then(function () {
        console.log('Success, You are a GEEK');
    }).
    catch(function () {
        console.log('Some error has occurred');
    });
```

Output:Success, You are a GEEK

Promise Consumers

Promises can be consumed by registering functions using .then and .catch methods.1. then() Promise Consumers

Promises can be consumed by registering functions using .then and .catch methods.1. then()
then() is invoked when a promise is either resolved or rejected. It may also be defined as a career which takes data from promise and further executes it successfully.Parameters:
then() method takes two functions as parameters. First function is executed if promise is resolved and a result is received.
Second function is executed if promise is rejected and an error is received. (It is optional and there is a better way to handle error using .catch() method
2. catch() catch() is invoked when a promise is either rejected or some error has occurred in execution. It is used as an Error Handler whenever at any step there is a chance of getting an error.