1. **What is a callback function?**

**Ans:** A callback, also known as a "call-after" function, is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action.

A synchronous callback is a function, that is executed immediately.

The callbacks, that are often used to continue code execution after an asynchronous operation has completed — these are called asynchronous callbacks.

1. **What is callback hell?**

**Ans:** Asynchronous JavaScript, or JavaScript that uses callbacks, is hard to get right intuitively. A lot of code ends up looking like the pyramid shape and all the ‘})’ at the end. This is affectionately known as **callback hell.**

The cause of callback hell is when people try to write JavaScript in a way where execution happens visually from top to bottom. Lots of people make this mistake! In other languages like C, Ruby or Python there is the expectation that whatever happens on line 1 will finish before the code on line 2 starts running and so on down the file.

1. **What are the advantages and disadvantages of callback funcitons.**

**Ans:**

**Advantage:** The callbacks, that are often used to continue code execution after an asynchronous operation has completed — these are called asynchronous callbacks.

**Disadvantage:** A synchronous callback is a function, that is executed immediately, which can be done simply without using callback.

1. **What is meant by asynchronous behaviour in javascript.**

**Ans:** In asynchronous programs, you can have two lines of code (L1 followed by L2), where L1 schedules some task to be run in the future, but L2 runs before that task completes.

**Note:** Asynchronous does not mean the same thing as concurrentor multi-threaded. **JavaScript can have asynchronous code, but it is generally single-threaded.**

1. **Explain how many ways we can create objects in javascript.**

**Ans:** There are different ways to create new objects:

* Define and create a single object, using an object literal.
* Define and create a single object, with the keyword new.
* Define an object constructor, and then create objects of the constructed type.

**Using an Object Literal**

* This is the easiest way to create a JavaScript Object.
* Using an object literal, you both define and create an object in one statement.
* An object literal is a list of name:value pairs (like age:50) inside curly braces {}.

**Example:** var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

## Using the JavaScript Keyword new

**Example:**

var person = new Object();  
person.firstName = "John";  
person.lastName = "Doe";  
person.age = 50;  
person.eyeColor = "blue";

For simplicity, readability and execution speed, use the first one (the object literal method).

## JavaScript Objects are Mutable

Objects are mutable: They are addressed by reference, not by value.

If person is an object, the following statement will not create a copy of person:

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"}  
var x = person;  
x.age = 10;

The object x is **not a copy** of person. It **is** person. Both x and person are the same object.

Any changes to x will also change person, because x and person are the same object.

1. **What is the cloning of object in javascript**

**Ans**: Cloning can be quite complex. Prototypal inheritance, reference types and methods associated with an object may require a specialized approach. Restrictions on cloned data may simplify cloning. It is the responsibility of the developers to understand and apply the correct cloning method on a case by case basis.

**Shallow Copy:** Cloning methods of most libraries are implemented using shallow copying. One example is \_.clone, the clone method of UnderscoreJs.

Shallow Copy: all field keys and values of the original object are copied to the new object.

**Deep Copy:**

When deeply cloning an object, all references are dereferenced. Only the structure of the object, key names and the atomic values are kept. A deep copy requires traversal of the whole object and building the cloned object from scratch.

1. **What is the difference between value types and reference types.**

**Ans:**

Javascript has 5 data types that are passed by value: Boolean, null, undefined, String, and Number. We’ll call these **primitive types**. If a primitive type is assigned to a variable, we can think of that variable as containing the primitive value.

Javascript has 3 data types that are passed by reference: Array, Function, and Object. These are all technically Objects, so we’ll refer to them collectively as **Objects**. Variables that are assigned a non-primitive value are given a reference to that value. That reference points to the object’s location in memory. The variables don’t actually contain the value.