Q: What is Event Driven Programming followed in Node.js?

In Event Driven Programming (EDP), the flow of a software program is based on the events, mostly user actions, like-Mouse clicks, key presses, sensor outputs.

EDP is mainly used in Graphic User Interface (GUI) projects.

In EDP, there is a main loop that keeps running and listening to events. As and when an event is detected, a callback function is triggered. This callback function is related to the event.

Q. What is a Callback function in Node.js?

A Callback function is an important part of Event driven programming in Node.js. We use Callabck functions to handle multiple requests made to the server. With a Callback function we can handle events in an Asynchronous way.

Let say we have a large list of numbers to sort. In Node.js we can pass this list to a function with the callback. The server can keep on handling the next request. Once the sort function finishes its work, it calls the callback function to inform server and emits an event. Based on this event further actions can be taken.

Q. How can we avoid Callback Hell in Node.js?

Node.js heavily depends on Callback functions. But at times developers create programs that create heavily nested Callbacks. This leads to unreadable spaghetti code that is difficult to comprehend.

We can use divide and conquer approach to avoid Callback Hell. We have to create loosely coupled modules in our Node.js program that handle specialized functions.

Q. What is Event Loop in Node.js?

Node.js is based on Event Driven programming model. At the heart of Node.js is an Event Loop. This Event Loop listens to events. On each event a call back function is called. This call back function is asynchronous in nature.

When we start Node.js, it internally starts the Event Loop.

Q. What is closure in JavaScript?

A Closure is an inner function that can access variables of outer function.

We use Closures extensively in Node.js. These are the main building blocks of asynchronous and non-blocking architecture of Node.js.

The inner function has access to not only its own variables but also to the parameters of outer function.

One simple way to create a closure is to include a function inside another function in JavaScript.

E.g.

```
function printName (name) {var address = “My name is ";

function addressMe () {return address +” “ + name;}

return addressMe ();}

printName(“James”); // My name is James
```

We even use Closures in Jquery also.

Q. What is the difference between Asynchronous and Non-blocking?

Asynchronous and non-blocking look very similar in nature but there is a subtle difference here.

Asynchronous means that our API will return immediately after calling it. In the background it will start a process to fulfill the request. Once that process is complete, our work is done. But our main thread does not wait for that process to complete.

Non-blocking means if our API cannot complete the work, it returns an error immediately. Based on the response, success of error, we can decide if we want to make another call to same API or continue with the next operation in our main flow. We can create a wait mechanism in a non-blocking operation.

Q. What is a Cluster in Node.js?

We can use cluster module to take advantage of multi-core system in a Node.js application.

We can create child processes that share server ports by using Cluster module.

These processes communicate via IPC.

Cluster module can distribute the incoming connections in Round Robin approach or in Interested Worker approach. In Round Robin approach, master distributes connections to workers in a round-robin fachion.

In Interested Worker approach, master distributes connections to interested workers only. Then workers can directly accept the connections.

Based on our application needs we can kill worker threads on need basis, without affecting other workers.

Q. What is a Child Process in Node.js?

There is a child_process module in Node.js that can be used to spawn child processes.

We can use commands like spawn(), exec(), fork() etc to create new Child Processes.

These child processes can be synchronous as well as asynchronous.

For shell script automation, synchronous child process can be used.

In asynchronous mode, we can specify a callback that will be called when Child process terminates.