- Node.js provides simplicity in development because of its non-blocking I/O and even-based model results in short response time and concurrent processing, unlike other frameworks where developers have to use thread management.

- It runs on a chrome v8 engine which is written in c++ and is highly performant with constant improvement.

- Also since we will use Javascript in both the frontend and backend the development will be much faster.

- And at last, there are ample libraries so that we don't need to reinvent the wheel.

5. Explain the steps how "Control Flow" controls the functions calls?

- Control the order of execution
- Collect data
- Limit concurrency
- Call the following step in the program.

6. What are some commonly used timing features of Node.js?

- **setTimeout/clearTimeout** – This is used to implement delays in code execution.
- **setInterval/clearInterval** – This is used to run a code block multiple times.
- **setImmediate/clearImmediate** – This is used to set the execution of the code at the end of the event loop cycle.
- **process.nextTick** – This is used to set the execution of code at the beginning of the next event loop cycle.

7. What are the advantages of using promises instead of callbacks?

So when an async function needs to be executed(or I/O) the main thread sends it to a different thread allowing v8 to keep executing the main code. Event loop involves different phases with specific tasks such as timers, pending callbacks, idle or prepare, poll, check, close callbacks with different FIFO queues. Also in between iterations it checks for async I/O or timers and shuts down cleanly if there aren't any.

18. If Node.js is single threaded then how does it handle concurrency?

The main loop is single-threaded and all async calls are managed by libuv library.

For example:

```
const crypto = require("crypto");
const start = Date.now();
function logHashTime() {
 crypto.pbkdf2("a", "b", 100000, 512, "sha512", () => {
console.log("Hash: ", Date.now() - start);
 });
}
logHashTime();
logHashTime();
logHashTime();
logHashTime();
```

This gives the output:

```
Hash: 1213
Hash: 1225
Hash: 1212
Hash: 1222
```

This is because libuv sets up a thread pool to handle such concurrency. How many threads will be there in the thread pool depends upon the number of cores but you can override this.

19. Differentiate between process.nextTick() and setImmediate()?

The main advantage of using promise is you get an object to decide the action that needs to be taken after the async task completes. This gives more manageable code and avoids callback hell.

8. What is fork in node JS?

A fork in general is used to spawn child processes. In node it is used to create a new instance of v8 engine to run multiple workers to execute the code.

9. Why is Node.js single-threaded?

Node.js was created explicitly as an experiment in async processing. This was to try a new theory of doing async processing on a single thread over the existing thread-based implementation of scaling via different frameworks.

10. How do you create a simple server in Node.js that returns Hello World?

```
var http = require("http");
http.createServer(function (request, response) {
  response.writeHead(200, {'Content-Type': 'text/plain'});
  response.end('Hello World\n');
}).listen(3000);
```

11. How many types of API functions are there in Node.js?

There are two types of API functions:

- **Asynchronous, non-blocking functions** - mostly I/O operations which can be fork out of the main loop.
- **Synchronous, blocking functions** - mostly operations that influence the process running in the main loop.

12. What is REPL?

PL in Node.js stands for **R**ead, **E**val, **P**rint, and **L**oop, which further means evaluating code on the go

Both can be used to switch to an asynchronous mode of operation by listener functions.

process.nextTick() sets the callback to execute but setImmediate pushes the callback in the queue to be executed. So the event loop runs in the following manner

timers–>pending callbacks–>idle,prepare–>connections(poll,data,etc)–>check–>close callbacks

In this process.nextTick() method adds the callback function to the start of the next event queue and setImmediate() method to place the function in the check phase of the next event queue.

20. How does Node.js overcome the problem of blocking of I/O operations?

Since the node has an event loop that can be used to handle all the I/O operations in an asynchronous manner without blocking the main function.

So for example, if some network call needs to happen it will be scheduled in the event loop instead of the main thread(single thread). And if there are multiple such I/O calls each one will be queued accordingly to be executed separately(other than the main thread).

Thus even though we have single-threaded JS, I/O ops are handled in a nonblocking way.

21. How can we use async await in node.js?

Here is an example of using async-await pattern:

```
const request = require('request');
const getPhotosByAlbumId = (id) => {
const requestUrl = `https://jsonplaceholder.typicode.com/albums/${id}/photos?_limit=3`;
return new Promise((resolve, reject) => {
    request.get(requestUrl, (err, res, body) => {
        if (err) {
            return reject(err);
        }
        resolve(JSON.parse(body));
    });
});
};
module.exports = getPhotosByAlbumId;
To test this function this is the stub
const expect = require('chai').expect;
const request = require('request');
const sinon = require('sinon');
const getPhotosByAlbumId = require('./index');
describe('with Stub: getPhotosByAlbumId', () => {
before(() => {
    sinon.stub(request, 'get')
        .yields(null, null, JSON.stringify([
            {
                "albumId": 1,
                "id": 1,
                "title": "A real photo 1",
                "url": "https://via.placeholder.com/600/92c952",
                "thumbnailUrl": "https://via.placeholder.com/150/92c952"
            },
            {
                "albumId": 1,
                "id": 2,
                "title": "A real photo 2",
                "url": "https://via.placeholder.com/600/771796",
                "thumbnailUrl": "https://via.placeholder.com/150/771796"
            },
            {
                "albumId": 1,
                "id": 3,
                "title": "A real photo 3",
                "url": "https://via.placeholder.com/600/24f355",
                "thumbnailUrl": "https://via.placeholder.com/150/24f355"
            }
        ]));
});
after(() => {
    request.get.restore();
});
it('should getPhotosByAlbumId', (done) => {
    getPhotosByAlbumId(1).then((photos) => {
        expect(photos.length).to.equal(3);
        photos.forEach(photo => {
            expect(photo).to.have.property('id');
            expect(photo).to.have.property('title');
            expect(photo).to.have.property('url');
        });
        done();
```