1. **Explain the use of JavaScript (or What you can do using a JavaScript)**

   JavaScript is a text-based programming language used both on the client-side and server-side that allows you to make web pages interactive. Where HTML and CSS are languages that give structure and style to web pages, JavaScript gives web pages interactive elements that engage a user. Common examples of JavaScript that you might use every day include the search box on Amazon, a news recap video embedded on The New York Times, or refreshing your Twitter feed.

   1. Adding interactive behavior to web pages
   2. Creating web and mobile apps
   3. Building web servers and developing server applications
   4. Game development

2. **What is the difference between client-side and server-side?**

   Client-side code, especially in the context of web development. Client-side code is code that is run on the user's computer — when a web page is viewed, the page's client-side code is downloaded, then run and displayed by the browser.

   Server-side code on the other hand is run on the server, then its results are downloaded and displayed in the browser. Examples of popular server-side web languages include PHP, Python, Ruby, and ASP.NET and... JavaScript! JavaScript can also be used as a server-side language, for example in the popular Node.js environment

3. **What is Node.js?**

   Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

4. Explain Scope in JavaScript

Scope in JavaScript refers to the current context of code, which determines the accessibility of variables to JavaScript. The two types of scope are local and global:

Global variables are those declared outside of a block

Local variables are those declared inside of a block

In the example below, we will create a global variable.

```
// Initialize a global variable
var creature = "wolf";
```

We learned that variables can be reassigned. Using local scope, we can actually create new variables with the same name as a variable in an outer scope without changing or reassigning the original value.

In the example below, we will create a global species variable. Within the function is a local variable with the same name. By sending them to the console, we can see how the variable's value is different depending on the scope, and the original value is not changed.

```
// Initialize a global variable
var species = "human";

function transform() {
  // Initialize a local, function-scoped variable
  var species = "werewolf";
  console.log(species);
}

// Log the global and local variable
console.log(species);
transform();
console.log(species);
```

```
Output

human

werewolf

human
```

In this example, the local variable is function-scoped. Variables declared with the `var` keyword are always function-scoped, meaning they recognize functions as having a separate scope. This locally-scoped variable is therefore not accessible from the global scope.

The new keywords `let` and `const`, however, are block-scoped. This means that a new, local scope is created from any kind of block, including function blocks, `if` statements, and `for` and `while` loops.

To illustrate the difference between function- and block-scoped variables, we will assign a new variable in an `if` block using `let`.

```
var fullMoon = true;

// Initialize a global variable
let species = "human";

if (fullMoon) {
  // Initialize a block-scoped variable
  let species = "werewolf";
  console.log(`It is a full moon. Lupin is currently a
${species}.`);
}

console.log(`It is not a full moon. Lupin is currently a
${species}.`);
```

```
Output

It is a full moon. Lupin is currently a werewolf.

It is not a full moon. Lupin is currently a human.
```

In this example, the `species` variable has one value globally (`human`), and another value locally (`werewolf`). If we were to use `var`, however, there would be a different result.

```
// Use var to initialize a variable
var species = "human";

if (fullMoon) {
  // Attempt to create a new variable in a block
  var species = "werewolf";
```

```
  console.log(`It is a full moon. Lupin is currently a
${species}.`);
}

console.log(`It is not a full moon. Lupin is currently a
${species}.`);
```

```
Output

It is a full moon. Lupin is currently a werewolf.

It is not a full moon. Lupin is currently a werewolf.
```

In the result of this example, both the global variable and the block-scoped variable end up with the same value, `werewolf`. This is because instead of creating a new local variable with `var`, you are reassigning the same variable in the same scope. `var` does not recognize `if` to be part of a different, new scope. It is generally recommended that you declare variables that are block-scoped, as they produce code that is less likely to unintentionally override variable values.

5. **JavaScript is asynchronous or synchronous.**

JavaScript is always synchronous and single-threaded. If you're executing a JavaScript block of code on a page then no other JavaScript on that page will currently be executed.

JavaScript is only asynchronous in the sense that it can make, for example, Ajax calls. The Ajax call will stop executing and other code will be able to execute until the call returns (successfully or otherwise), at which point the callback will run synchronously. No other code will be running at this point. It won't interrupt any other code that's currently running.

JavaScript timers operate with this same kind of callback.

Describing JavaScript as asynchronous is perhaps misleading. It's more accurate to say that JavaScript is synchronous and single-threaded with various callback mechanisms.

6. **JavaScript is Single-threaded or Multi-threaded.**

JavaScript is a single-threaded language because while running code on a single thread, it can be really easy to implement as we don’t have to deal with the complicated scenarios that arise in the multi-threaded environment like deadlock.

Since, JavaScript is a single-threaded language, it is synchronous in nature.

7. **Explain DOM in your own word.**

The Document Object Model (DOM) is a programming interface for web documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as nodes and objects; that way, programming languages can interact with the page.

A web page is a document that can be either displayed in the browser window or as the HTML source. In both cases, it is the same document but the Document Object Model (DOM) representation allows it to be manipulated. As an object-oriented representation of the web page, it can be modified with a scripting language such as JavaScript.