Explain differences between Java and Javascript:

- Compile-time errors will not show in javascript, and will only show on runtime, therefore better code completion.
- Languages that are compiled are normally faster.
- JS is a bit more flexible in terms of syntax as compared to java which is very strict.
- JS doesn't have typecheck.
- JS is focused around functions where java is focused on objects

- Difference in object creation:
- JavaScript: Var a = {} OR Var b = new Person()
- Java: Object objectName = new Object();

Explain using sufficient code examples the following features in JavaScript:

- **Use strict**

It is not a statement, but a literal expression, ignored by earlier versions of JavaScript.
The purpose of "use strict" is to indicate that the code should be executed in "strict mode".
With strict mode, you cannot, for example, use undeclared variables.

Using the Strict-mode in JavaScript is a way to keep code clean by having higher standards than usual. This can be a really good thing, since JavaScript is a very forgiving language; it allows you to not put a semi-colon here and there, and as a programmer, you can get very sloppy when coding. Strict mode means for instance that you MUST declare variables before you use them. If that has not been done while in strict mode, the script will break and will cause an actual error.

Usually JavaScript also allows you to name the function parameters however you like, but in strict mode this is NOT the case. Each parameter must have a differentname.

One thing about strict mode is that not all browswers support it, and if you introduce strict mode in one .js file, you must use it in ALL of the .js files.

```
function foo() {
name = "John";
}
var name = ""; //OK with strict mode OFF.
//------------------BUT WILL HAVE TO LOOK LIKE THIS WITH STRICT MODE ON--------
"use strict";
var name;
function fooIt() {
name = "John";
}
*/
```

- **Variable/function-hoisting**

Variable-hoisting
JS is a bit different than other languages, which means that it can detect whether a variable just hasn't been declared yet or not at all. JS "looks ahead" through the rest of the code, to see if the variable in question has been declared.
If the variable isn't declared at all it will output "variableName is not defined".
If the variable hasn't been declared **yet** it will output "variableName: undefined".

Function-hoisting:
A function declaration doesn't just hoist the function's name. It also hoists the actual function definition. So you call a function before it's declared. However, **function definition hoisting only occurs for function declarations**, not function expressions.

- **This in JavaScript and how it differs from what we know form Java/.net**

This is a bit tricky in JavaScripts, but here goes:
Whenever a function is contained in the global scope, the value of this inside of that function will be the global object (window in a browser) or undefined if in strict mode.
Whenever a function is called by a preceding dot, the object before that dot is this.
Whenever a constructor function is used, this refers to the specific instance of the object that is created and returned by the constructor function.
Whenever JavaScript's call or apply method is used, this is explicitly defined.

'this' in JAVA:
The this-variable in Java is dependent on class instantiation; whenever there is a class instantiation (an object is made), a 'this' is created and refers to that very object and ONLY this object when used.

'this' in JavaScript:
The this-variable in JavaScript refers to different things / objects depending on how it's called.
ie. is the 'this' is called from outside any function the 'this' refers to the global object. - if the 'this' is called from within a function it depends on how the function is written:

- **IIFE**

IIFE's can be used to avoid variable hoisting from within blocks, protect against polluting the global environment and simultaneously allow public access to methods while retaining privacy for variables defined within the function.

A common way to implement IIFE's is to enclose both the function expression and invocation in parentheses.

- **Use the debugger to explain the basic "things" all objects inherits from object**
- **User defined callback functions.**

Functions that takes a callback (another function) as an argument, are known as callback functions.