

## Period-1 Vanilla JavaScript, es2015/15., Node.js, Babel + Webpack and TypeScript



Note: This description is too big for a single exam-question. It will be divided up into a number of smaller questions for the exam

Explain and Reflect:


- Explain the differences between Java and JavaScript and Java and node. Topics you could include:
  - that Java is a compiled language and JavaScript a scripted language
  - Java is both a language and a platform
  - General differences in language features.
  - Blocking vs. non-blocking
- Explain generally about node.js, when it “makes sense” and *npm*, and how it “fits” into the node echo system.
- Explain about the Event Loop in JavaScript, including terms like; blocking, non-blocking, event loop, callback queue and "other" API's. Make sure to include why this is relevant for us as developers.
  - Link to eventloop video <https://www.youtube.com/watch?v=8aGhZQkoFbQ>
- Explain the terms JavaScript Engine (name at least one) and JavaScript Runtime Environment (name at least two)
  - V8 is the JS engine chrome uses
  - NODE.js is a javascript runtime environment used which can run server and desktop apps.
- Explain (some) of the purposes with the tools *Babel* and *WebPack* and how they differ from each other. Use examples from the exercises.
  - Babel is used to translate the new hip code to older versions of JS that browsers which run on later versions can read. Ie, it can translate from ES6 to ES5
  - Webpack often include babel as one of its “tools/jobs” but is also used to bundle up code so it becomes more smooth, and managed in an orderly fashion.

Explain using sufficient code examples the following features in JavaScript (and node)

- Variable/function-Hoisting
  - Hoisting is when the variable or function that is created, is hoisted “to the top” of the code. Ie. If a “var” is created somewhere in the middle of the code, the variable itself is hoisted to the top (just under functions) but is only declared at the point in the code were it is actually written.
- *this* in JavaScript and how it differs from what we know from Java/.net.

TAKEN FROM W3SCHOOLS



- It has different values depending on where it is used:
- In a method, **this** refers to the **owner object**.
- Alone, **this** refers to the **global object**.
- In a function, **this** refers to the **global object**.

- In a function, in strict mode, **this** is **undefined**.
  - In an event, **this** refers to the **element** that received the event.
  - Methods like **call()**, and **apply()** can refer **this** to **any object**.
  - Link to source [https://www.w3schools.com/js/js\\_this.asp](https://www.w3schools.com/js/js_this.asp)
-  Function Closures and the JavaScript Module Pattern
- User-defined Callback Functions (writing your own functions that take a callback)
  - A callback 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.

```

1  function greeting(name) {
2      alert('Hello ' + name);
3  }
4
5  function processUserInput(callback) {
6      var name = prompt('Please enter your name. ');
7      callback(name);
8  }
9
10 processUserInput(greeting);

```

- Explain the methods `map`, `filter` and `reduce`
  - The `map` method loops through an array and “does something” on every item
  - The `filter` method filters through the array (ie. Find all people above age 18)
-  Provide examples of user-defined reusable modules implemented in Node.js (learnynode - 6)
- Provide examples and explain the es2015 features: `let`, arrow functions, `this`, `rest` parameters, destructuring objects and arrays,  `maps/sets` etc.
  - `let` is used to create local variables
  - Arrow functions is a compact way to create anonymous functions
  - The **rest parameter** syntax allows us to represent an indefinite number of arguments as an array.

```


1  function sum(...theArgs) {
2      return theArgs.reduce((previous, current) => {
3          return previous + current;
4      });
5  }
6
7  console.log(sum(1, 2, 3));
8  // expected output: 6
9
10 console.log(sum(1, 2, 3, 4));
11 // expected output: 10
12

```


- Destructuring is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables

```
var introduction = ["Hello", "I", "am", "Sarah"];
var [greeting, pronoun] = introduction;

console.log(greeting); //"Hello"
console.log(pronoun); //"I"
```


- **Map** is a collection of keyed data items, just like an **Object**. But the main difference is that Map allows keys of any type.
- A **Set** is a special type collection – “set of values” (without keys), where each value may occur only once.  
Source: <https://javascript.info/map-set>
-  Provide an example of ES6 inheritance and reflect over the differences between Inheritance in Java and in ES6.
- Explain and demonstrate, how to implement your own events, how to emit events and how to listen for such events
  - See week2 dosdetector project.

#### ES6,7,8,ES-next and TypeScript

- Provide examples with es-next, running in a browser, using Babel and Webpack
- Explain the two strategies for improving JavaScript: Babel and ES6 (es2015) + ES-Next, versus Typescript. What does it require to use these technologies: In our backend with Node and in (many different) Browsers
  - Use webpack and babel to bundle up files and eliminate the “generation gap”
  - Use typescript as a higher level of javascript abstraction which also handles translation to eliminate “generation gap”
- Provide a **number of examples** to demonstrate the benefits of using TypeScript, including, types, interfaces, classes and generics
  - See week5 projects
-  Explain the ECMAScript Proposal Process for how new features are added to the language (the TC39 Process)


#### Callbacks, Promises and async/await

Explain about (ES-6) promises in JavaScript including, the problems they solve, a quick explanation of the Promise API and:

- See week3 project.
-  Example(s) that demonstrate how to avoid the callback hell (“Pyramid of Doom”)
- Example(s) that demonstrate how to execute asynchronous (promise-based) code in **serial** or **parallel**
- Example(s) that demonstrate how to implement **our own** promise-solutions.
- Example(s) that demonstrate error handling with promises

Explain about JavaScripts **async/await**, how it relates to promises and reasons to use it compared to the plain promise API.

Provide examples to demonstrate

- Why this often is the preferred way of handling promises
- Error handling with async/await
-  Serial or parallel execution with async/await.

Se the exercises for Period-1 to get inspiration for relevant code examples