Do you know the output of the following?

1.

```
const arr = [10, 12, 15, 21];

for (let i = 0; i < arr.length; i++) {
  setTimeout(function() {
    console.log('Index: ' + i + ', element: ' + arr[i]);
  }, 3000);
}
```

2.

```
function a(foo) {
  if (foo > 20) {
    return foo;
  }

  return b(foo + 2);
}

function b(foo) {
  return c(foo) + 1;
}

function c(foo) {
  return a(foo * 2);
}

a(1);
```

BendJS

1.

```
4 Index: 4, element: undefined
>
```

> *The reason for this is because the* `setTimeout` *function creates a function (the closure) that has access to its outer scope, which is the loop that contains the index* `i`*. After 3 seconds go by, the function is executed and it prints out the value of* `i`*, which at the end of the loop is at 4 because it cycles through 0, 1, 2, 3, 4 and the loop finally stops at 4.* `arr[4]` *does not exist, which is why you get* `undefined`*.*

2. 39

Sources: https://medium.com/coderbyte/a-tricky-javascript-interview-question-asked-by-google-and-amazon-48d212890703

BendJS

BendJS

Getting started with TypeScript

BendJS - 2017-12-12

Adam Horak

Solution Architect @

frontline education™

What is TypeScript?

- Superset of JavaScript with static typing
- Includes some future JS features (dynamic import, decorators, etc)
- Compiles to plain JavaScript (configurable targets based on ES5, ES6, etc)
- Plain old JavaScript is valid TypeScript!

Why?

- Catches code that is not type safe before it is run and eliminates some hidden bugs
- Communicates type information that could otherwise be implicit to other developers
- Reminds future me of WTF I was thinking when I wrote something 6 months ago
- Imposes discipline in design
- Refactoring!

Why not?

- Extra friction for setup and compilation
- Unnecessary overhead for prototyping or small projects
- More syntax to learn and concepts to understand

Setting up a TypeScript project

- echo "{}" >> tsconfig.json && touch index.ts
- done!

Basic type annotations and type inference

- const foo: string = “foo”;
- const sayHello = (name: string): string => `Hello ${name}`
- const sayHello = function(name: string): string {...}
- Basic types:
  - boolean
  - number
  - string
  - array
  - tuple
  - null
  - undefined
- Inline type annotations

Interfaces

- Define object shapes and member types
- Define function parameters + return types
- Optional properties
- Readonly properties
- Index signatures
- Extending interfaces

Type aliases

- “type” keyword declares a type that can be re-used elsewhere
- You can declare a type alias with any type annotation

Union and intersection types

- Union types declare an “or” relationship
- Intersection types declare and “and” relationship

Escape hatches

- Any type
- Type assertions

Generics

- Type variables
- function identity<T> (arg: T): T { return arg; }
- Often inferred
- Generic constraints
- Examples:
  - Promise<number>
  - Array<string>

Enums

- Numeric enums
- String enums
- Run-time code

Configuring compilation behavior

- tsconfig.json
- noImplicitAny
- strictNullChecks

Third party libraries

- DefinitelyTyped
- npm install @types/express

A (very) simple example

- Type safe Express API

Community Poll - Meetup Website Domain Suffix

- Bendjs.io
- Bendjs.com
- Bendjs.org
- Bendjs.casino
- Bendjs.diamonds
- Bendjs.dentist
- Bendjs.lol

BendJS