Cracking the JavaScript interviews

Based on experience

ABOUT ME

Albert Hovhannisian

Co-Founder & CTO @SharpIdea

albert.hovhannisian.main@gmail.com

medium - @AlbertHovhannisian

BEFORE WE START

Agenda

01. The interview process
02. Behavioral questions
03. Problem solving flow
04. Offer
05. Summary/advices

INTERVIEWS IN GENERAL

06. Types of technical interviews
07. Interview questions (explained)
08. Interview problems (solved)
09. Big O Notation

TECHNICAL

10. What we get ?
11. Question of the era (advices for beginners)
13. Q & A

FINAL WORDS

12. Challenge

WHY ?

01. THE INTERVIEW PROCESS

01. ANALYTICAL THINKING

Did you need help to solve the issue ?
Was your solution optimized ?
Is the structure of solution reusable ?

02. CODING SKILLS

Were you able to translate the algorithm to the code ?
Is the code well organized ?
Did you thought about the corner cases and error ?

03. TECHNICAL KNOWLEDGE

Did you know the basic of computer science ?

04.EXPERIENCE

Have you made good technical decisions before ?
Have you built challenging projects ?

05. SOFT SKILLS

Do your personality and values fit with the company and team ?
Did you communicate well with your interviewer ?

01.2 THE INTERVIEW PROCESS

SCREENING INTERVIEW

- It can contain coding/algorithmic questions as well
- If you're not sure whether or not the interview will be technical, you can always ask your recruiting coordinator what position your interviewer holds

DEPEND ON COMPANY THERE CAN BE

- Synchronized document where you should describe some stuff (code, algorithm, etc...)
- Home tasks after screening

ALWAYS EXPLORE THE COMPANY / PRODUCT

04. BEHAVIORAL QUESTIONS

Qommon questions	Project 1	Project 2	Project 3
Challenges			
Mistakes			
Benefits / what you enjoyed			
Leadership			
Conflicts			
What you'd do differently			

04.1 BEHAVIORAL QUESTIONS

What are your weaknesses ?

Give a real weakness

04.1 WHAT QUESTIONS YOU SHOULD ASK TO INTERVIEWER ?

Genuine questions (you actually want to know)

Insightful Questions (demonstrate your knowledge)

Passion questions (show your passiong to technology)

04.2 TIPS FOR RESPONDING

01. BE SPECIFIC
02. LIMIT DETAILS IN YOUR ANSWER
03. TELL ABOUT YOURSELF (FOCUS ON YOURSELF)
04. GIVE STRUCTURED ANSWERS
    04.1 NUGGET FIRST
    04.2 S.A.R.

06. PROBLEM SOLVING FLOW

Listen carefully and ask questions	**Visualize the problem**	**Divide it into parts**	**Brute force**
Optimize	**Walk trough**	**Implement**	**Test**

! Always keep talking

07. OFFER

01. OFFER HAS DEADLINES
02. DECLINE THE OFFER
03. HANDLE THE REJECTION
04. EVALUATE THE OFFER

SUMMARY/ADVICES

1. ALWAYS EXPLORE THE COMPANY / PRODUCT
2. BE PREPARED FOR BEHAVIORAL QUESTIONS
3. PREPARE AND ASK QUESTIONS
4. BRUSH UP YOUR SKILLS BEFORE INTERVIEW
5. CHOOSE SOME OF THE PROJECTS THAT YOU BUILT AND BE READY FOR PRESENTING THEM (MOST CHALLENGING, AND THE LAST ONE)
6. FOLLOW TO PROBLEM-SOLVING FLOW CHART

PART II

TECHNICAL QUESTIONS

05. TECHNICAL INTERVIEW

MOSTLY THEORETICAL	MOSTLY PRACTICAL (ALGORITHMIC)
Questions about language features	Here can be only one algorithmic problem
Some basic questions about computer science	
Basic algorithmic task	

05. MOSTLY THEORETICAL

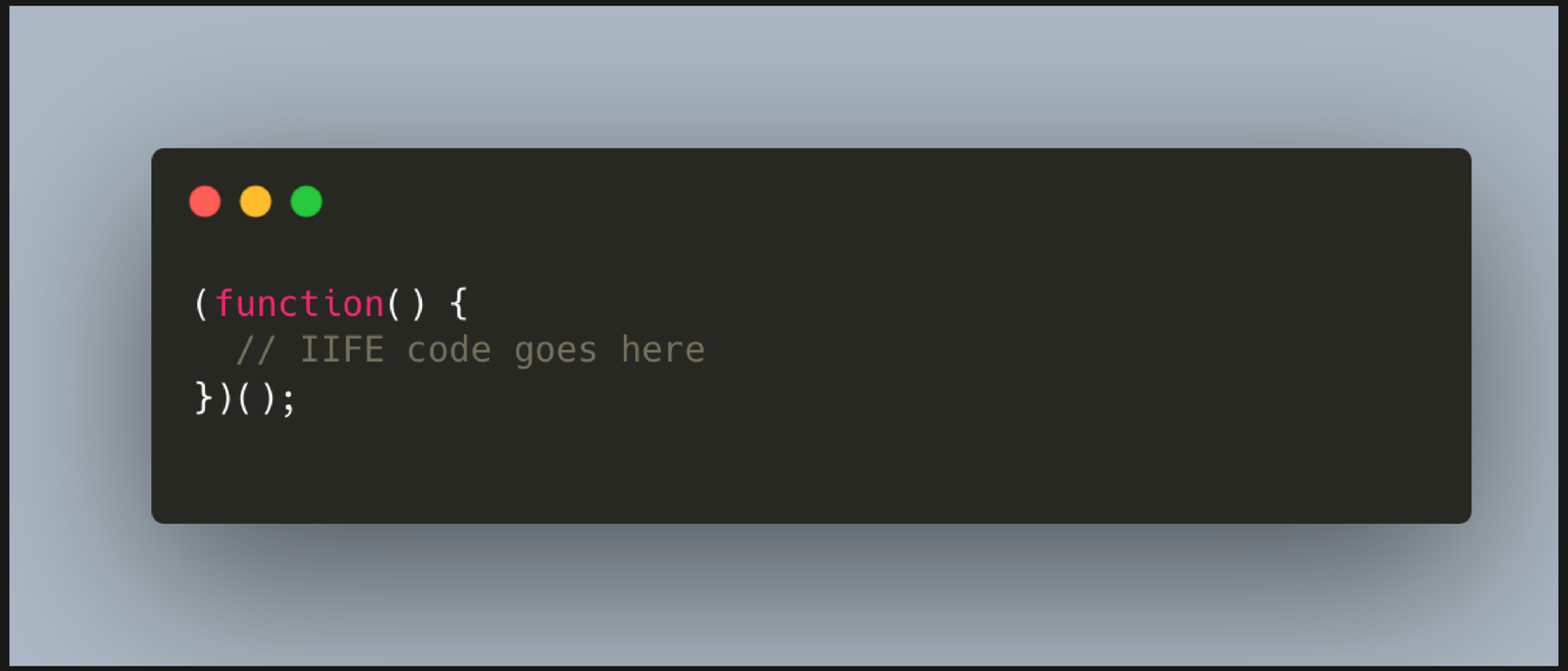
1. Core JavaScript concepts: variables, data types, operators, functions, arrays, and objects.
2. DOM manipulation and events.
3. Asynchronous programming
4. Frameworks and libraries
5. Performance optimization
6. Testing and debugging
7. Security
8. Basic algorithmic tasks

Q1. EXPLAIN THE DIFFERENCE BETWEEN OBJECT.FREEZE() AND CONST

const and Object.freeze are two completely different things.
const applies to bindings ("variables"). It creates an immutable binding
Object.freeze works on values, and more specifically, object values. It makes an object immutable.

Q2. WHAT ARE IIFE'S AND WHAT IS THE USE CASE ?

Immediately Invoked Function Expression



```
(function() {
  // IIFE code goes here
})();
```

Creates new scope

Q3. .MAP() VS .FOREACH()

```
const numbers = [1, 2, 3, 4, 5];

// Using map() to create a new array with doubled values
const doubledNumbers = numbers.map(num => num * 2);
console.log(doubledNumbers); // [2, 4, 6, 8, 10]

// Using forEach() to log each number to the console
numbers.forEach(num => console.log(num));

// Using forEach() to modify the original array
numbers.forEach((num, index) => numbers[index] = num *
2);
console.log(numbers); // [2, 4, 6, 8, 10]
```

Q4. EXPLAIN HOISTING IN JS

All variable declarations (with var) and all function definitions are hoisting up to the start of the parent context

```
console.log(myVar); // undefined
var myVar = 42;

sayHello(); // "Hello, world!"
function sayHello() {
  console.log("Hello, world!");
}
```

Q5. WHAT IS THE OUTPUT OF THE FOLLOWING CODE ?

```
console.log("A");
setTimeout(function() {
  console.log("B");
  setTimeout(function() {
    console.log("C");
  }, 0);
}, 1000);
setTimeout(function() {
  console.log("D");
}, 500);
```

Q5. WHAT IS THE DIFFERENCE BETWEEN MICROTASKS AND MACROTASKS IN THE EVENT LOOP?

Q6. WHAT'S THE OUTPUT ?

```
console.log("1");

setTimeout(function() {
  console.log("2");
  Promise.resolve().then(function() {
    console.log("3");
  });
});

Promise.resolve().then(function() {
  console.log("4");
  setTimeout(function() {
    console.log("5");
  });
});

console.log("6");
```

05. PROBLEM

Find the longest common prefix in array of strings

```
longestCommonPrefix(['flower', 'floor']) // "flo"
longestCommonPrefix(['flower', 'floor', 'flat']) // "fl"
```

05. SOLVING

Step 1: Visualizing the problem

flower floor flat

o === o !== a => break;

Step 2: Divide into parts

Iterate trough all elements

Make character checking

Step 3: Brute force

step 1: get base

step 2: compare the base with other values

step 3: "Generate" the prefix value

step 4: Return the prefix value

Step 4: Optimize

Step 5: Implement

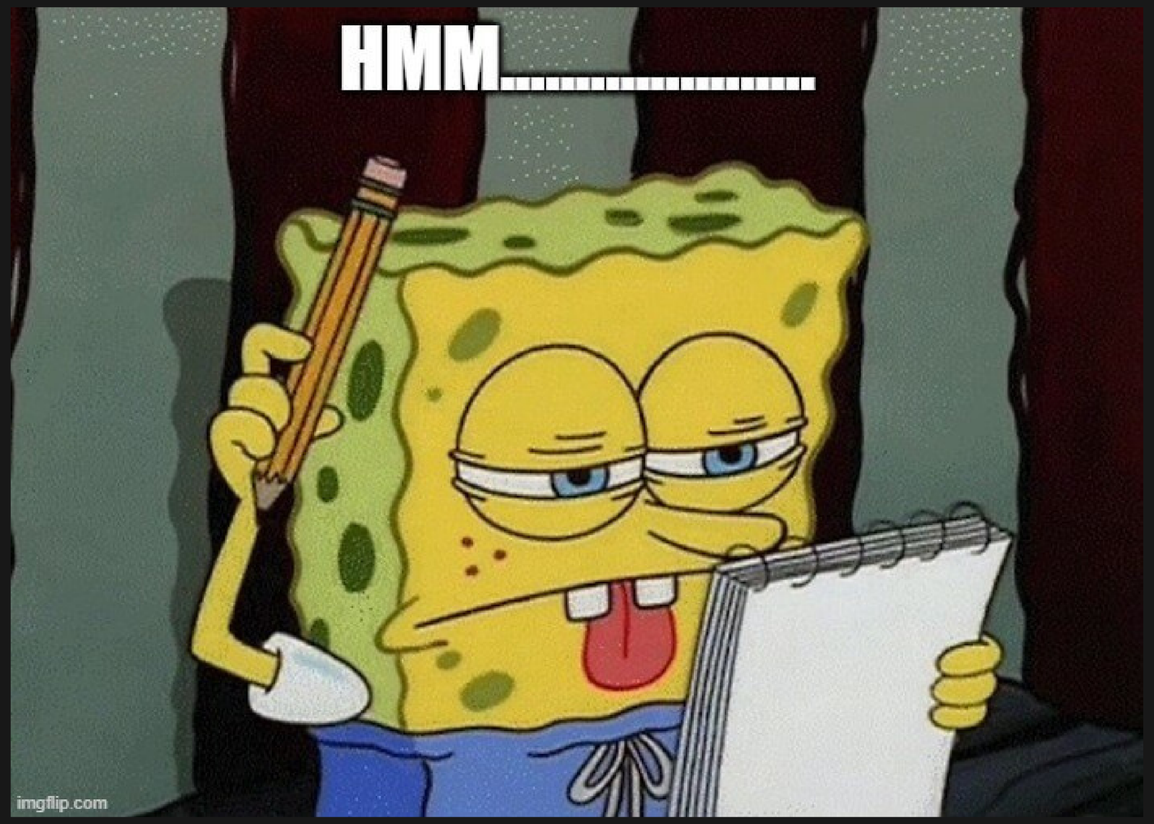
05. IMPLEMENTATION

```
function longestCommonPrefix(strs) {
  let prefix = strs[0];
  for (let i = 1; i < strs.length; i++) {
    while (strs[i].indexOf(prefix) !== 0) {
      prefix = prefix.substring(0, prefix.length - 1);
      if (prefix === "") return "";
    }
  }
  return prefix;
}
```

05. RESULT

```
> function longestCommonPrefix(strs) {
    let prefix = strs[0];
    for (let i = 1; i < strs.length; i++) {
      while (strs[i].indexOf(prefix) !== 0) {
        prefix = prefix.substring(0, prefix.length - 1);
        if (prefix === "") return "";
      }
    }
    return prefix;
  }
< undefined
> longestCommonPrefix(['flower', 'flat', 'fly'])
< 'fl'
>
```

05. DID WE FINISHED THE ALGORITHM ?

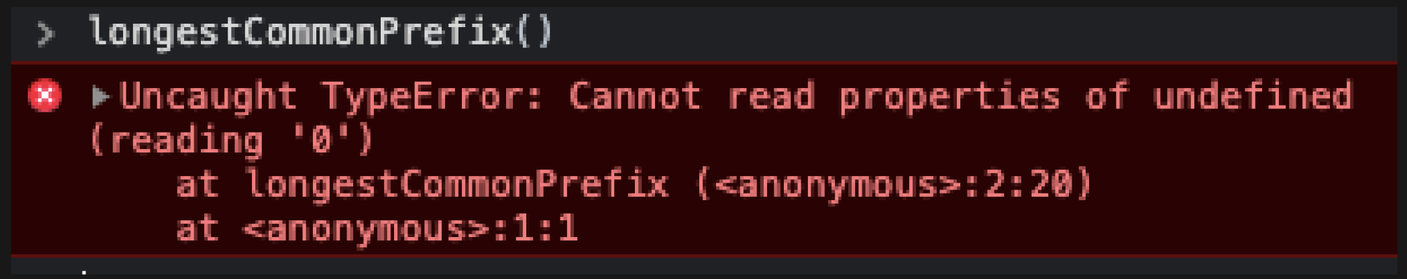


Nope, actually we have missed a step

Testing !

05. ISSUE

What if there are no any inputs ?

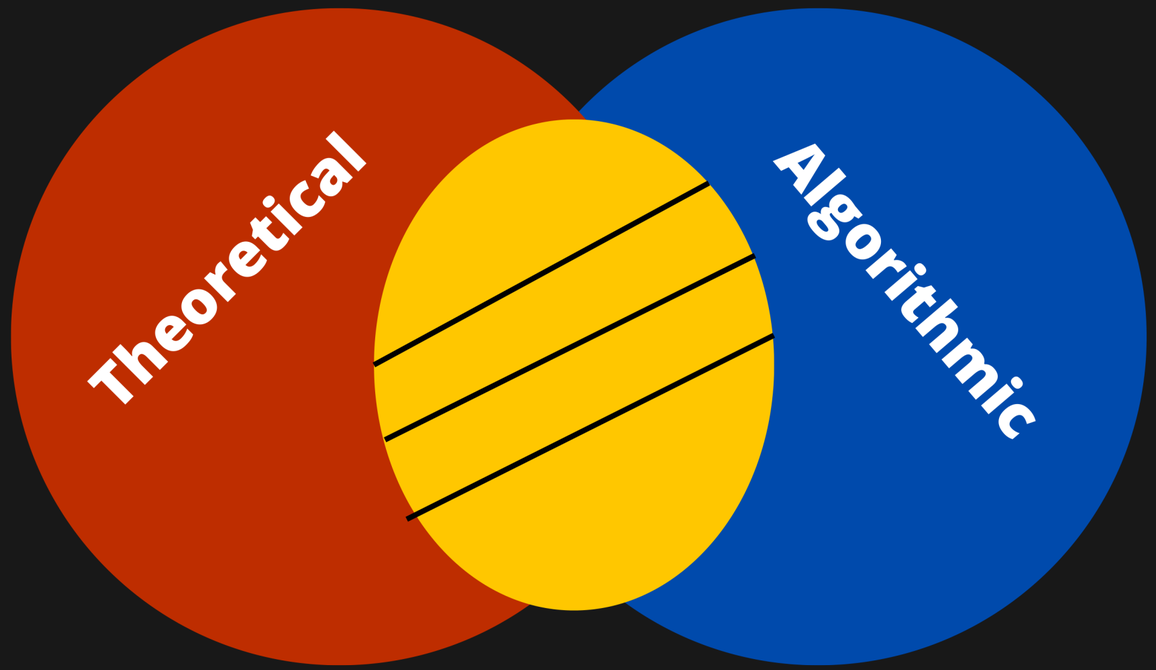


Here we missed to validate the input

05. SOLUTION

```javascript
function longestCommonPrefix(strs) {
  if (strs.length === 0) return "";
  let prefix = strs[0];
  for (let i = 1; i < strs.length; i++) {
    while (strs[i].indexOf(prefix) !== 0) {
      prefix = prefix.substring(0, prefix.length - 1);
      if (prefix === "") return "";
    }
  }
  return prefix;
}
```

05. CROSSING



< = > **Big O**

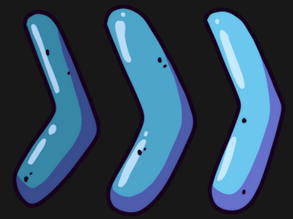
05.ALGORITHMS AND BIG O

AN ALGORITHM

Sequence of instructions for solving **clearly defined problem**

The same steps will lead to the same output

Every program is an algorithm



As a programmer you should be able to solve problems

05. WHAT MEANS TO FIND THE BEST SOLUTION ?

Minimum amout of code ?

Better performance?

Less memory usage?

Personal preferences ?

05. WHAT MEANS TO FIND THE BEST SOLUTION ?

Minimum amout of code ?

Better performance?

Less memory usage?

Personal preferences ?

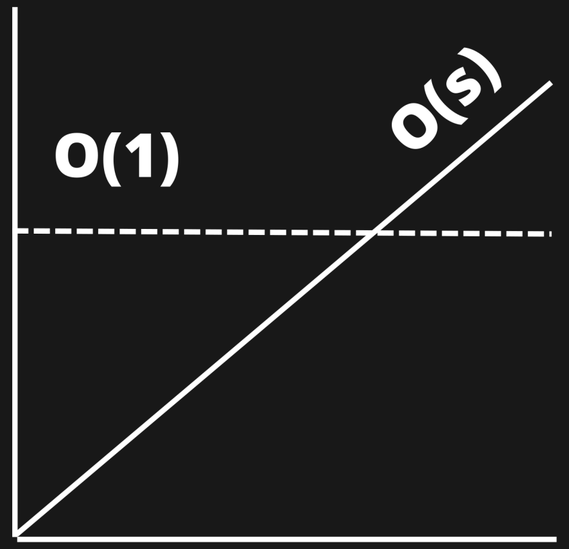
05. MEASURING PERFORMANCE
BIG O

USED TO DESCRIBE THE EFFICIENCY OF ALGORITHM

Time complexity **Space complexity**

REAL LIFE EXAMPLE

1. **Electronic transfer O(s)**
2. **Airplane transfer O(1)**

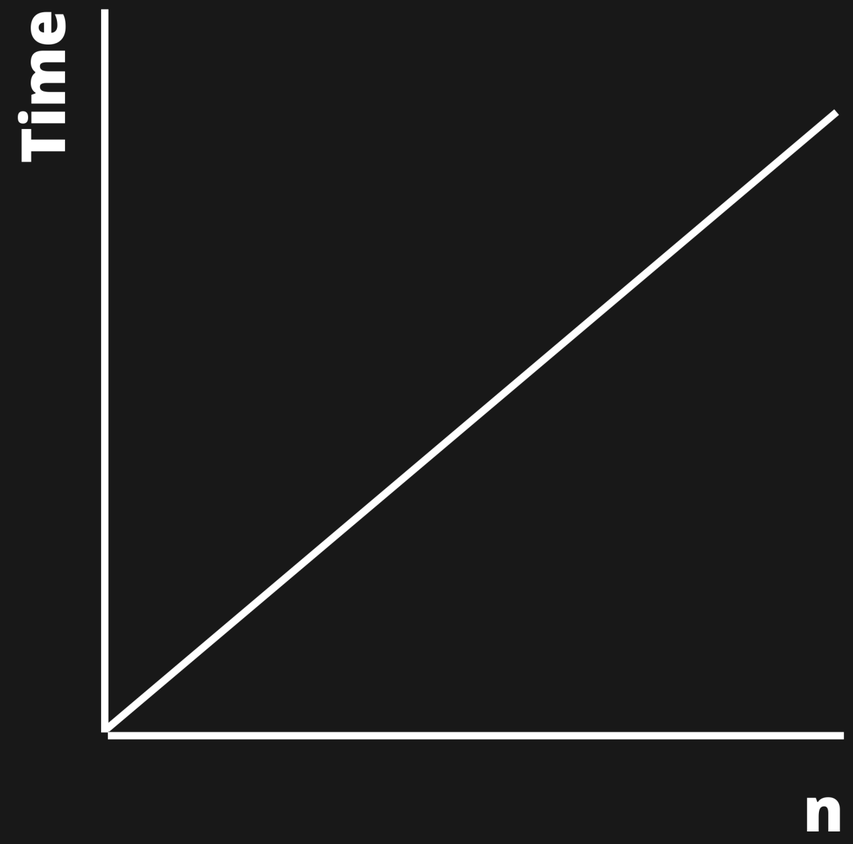


05.EXAMPLE WITH JS

```js
const calculateSum = (n) => {
  let result = 0;

  for(let i = 1; i <= n; i++) {
    result += i;
  }

  return result;
}

calculateSum(3) // 6
calculateSum(4) // 10
```

```
> start = performance.now()
  calculateSum(100000)
  end = performance.now()
  end - start
< 0.30000001192092896

> start = performance.now()
  calculateSum(1000000)
  end = performance.now()
  end - start
< 2.199999988079071

> start = performance.now()
  calculateSum(10000000)
  end = performance.now()
  end - start
< 14.800000011920929
```

05.EXAMPLE WITH JS



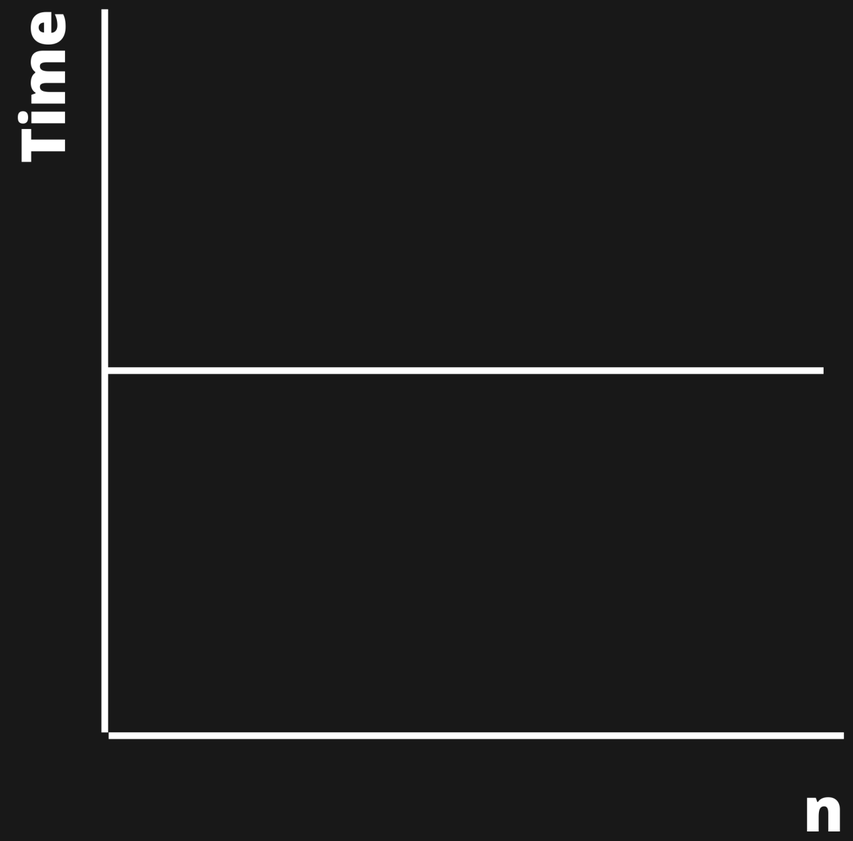
=> More loop iterations => time increases in a linear way

CAN WE OPTIMIZE THIS ?

```
const calculateSum = (n) => {
  return (n / 2) * (1 + n);
}

calculateSum(3) // 6
calculateSum(4) // 10
```

```
> start = performance.now()
  calculateSum(100000)
  end = performance.now()
  end - start
< 0.09999996423721313

> start = performance.now()
  calculateSum(1000000)
  end = performance.now()
  end - start
< 0

> start = performance.now()
  calculateSum(10000000)
  end = performance.now()
  end - start
< 0
```

05.EXAMPLE WITH JS



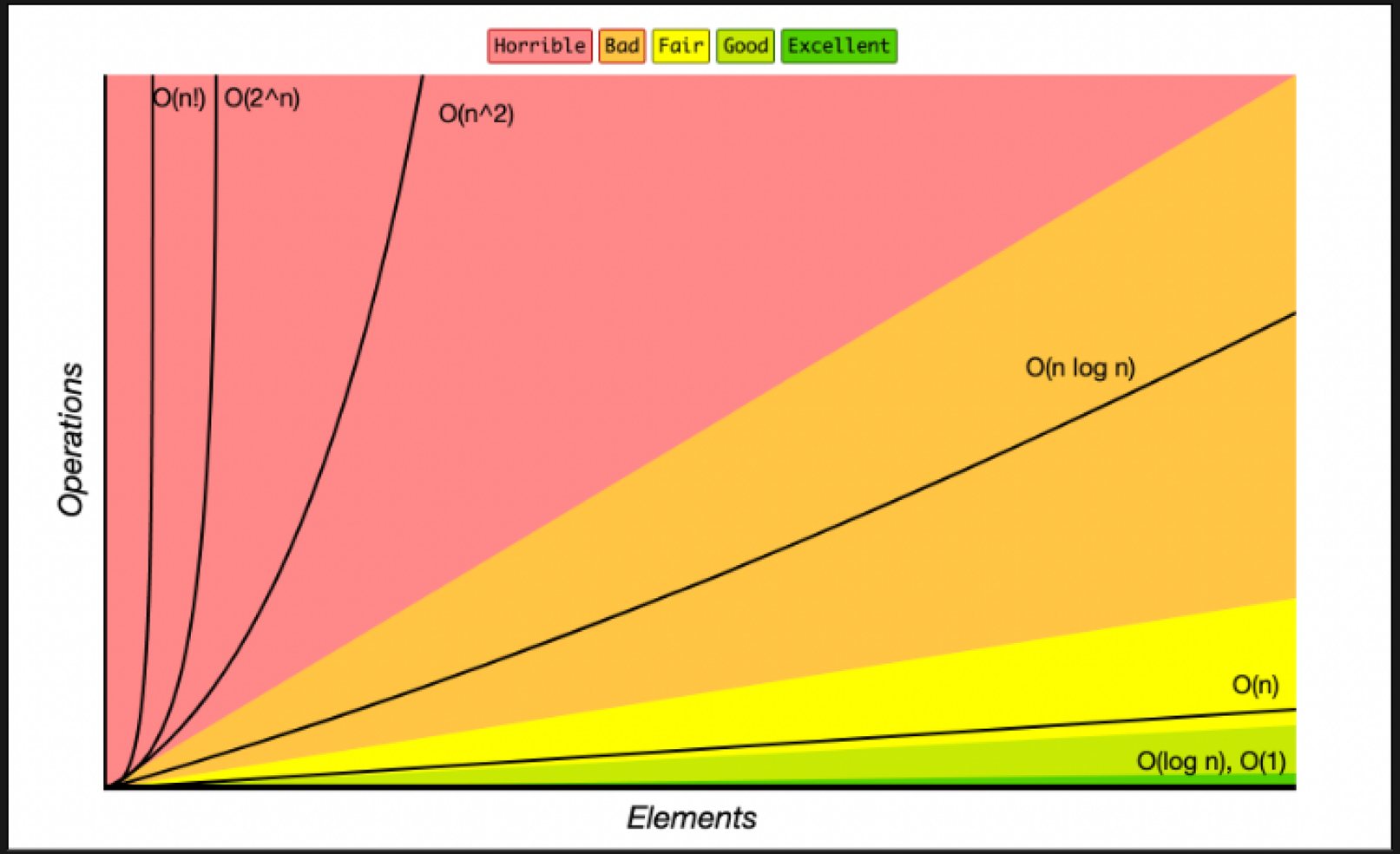
=>

No iterations, no loops => constant time

05 THE BIG O

O(n) - linear time complexity
O(1) - constant time complexity
O(n^2) - Quadratic time complexity
etc...

FUNCTION VIEW: MATHEMATICAL



HOW TO CALCULATE THE COMPLEXITY OF AN ALGORITHM ?

STEP I: WE NEED TO COUNT HOW MANY TIMES THE SINGLE LINE OF CODE IS GETTING EXECUTED WHEN THE INPUT DATA IS GROWING

```
const calculateSum = (n) => {
  let result = 0; O(1)

  for(let i = 1; i <= n; i++) { O(1)
    result += i; O(n)
  }

  return result; O(1)
}

calculateSum(3) // 6
calculateSum(4) // 10
```

O(1 + 1 + n + 1) =
O (3 + n)

O (~~3~~ + n) = O(n)

=> linear complexity

HOW TO CALCULATE THE COMPLEXITY OF AN ALGORITHM ?

01.

Remove constants

02.

Find the fastest growing part

03.

Produce rule

Examples

O(4n) = ?

O(n/4) = ?

O(n^2 + 1 + n + n^2) = ?

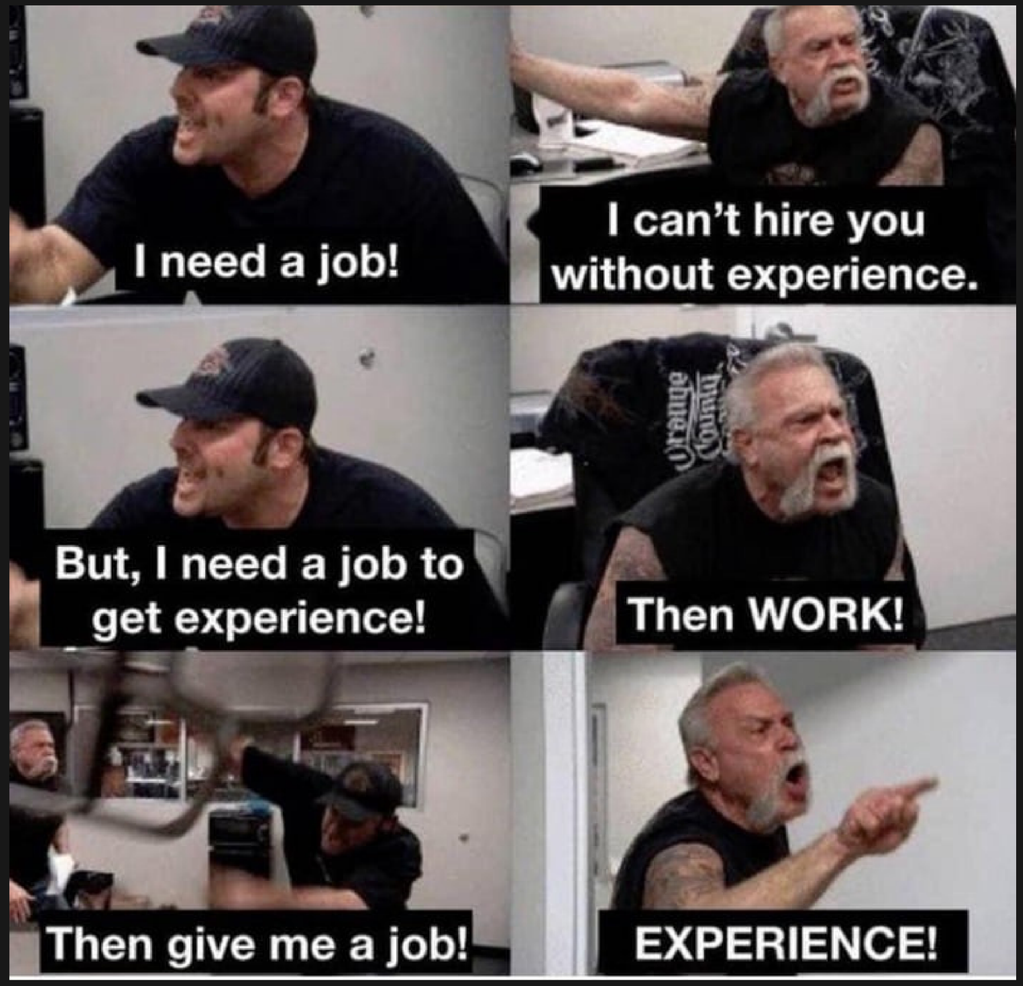
PART III

FEW WORDS

09.WHAT WE GET ?

01. HOW THE INTERVIEW PROCESSES ARE BUILT ?
02. TIPS FOR RESPONDING TO BEHAVIORAL QUESTIONS
03. PROBLEM-SOLVING FLOW
04. HANDLING THE OFFERS
05. ALGORITHMS AND BIG O
06. CALCULATING THE EFFICIENCY OF ALGORITHMS
07. COMPARING ALGORITHMS
08. SOME POPULAR INTERVIEW QUESTIONS (EXPLAINED)
08. SOME POPULAR INTERVIEW PROBLEMS (EXPLAINED)

10.QUESTIONS OF THE ERA



10.QUESTIONS OF THE ERA

01. BUILD A STRONG PORTFOLIO

02. FOLLOW TO JOB POSTING GROUPS

03. APPLY FOR ENTRY LEVEL POSITIONS

04. CONTRIBUTE TO OPEN SOURCE PROJECTS

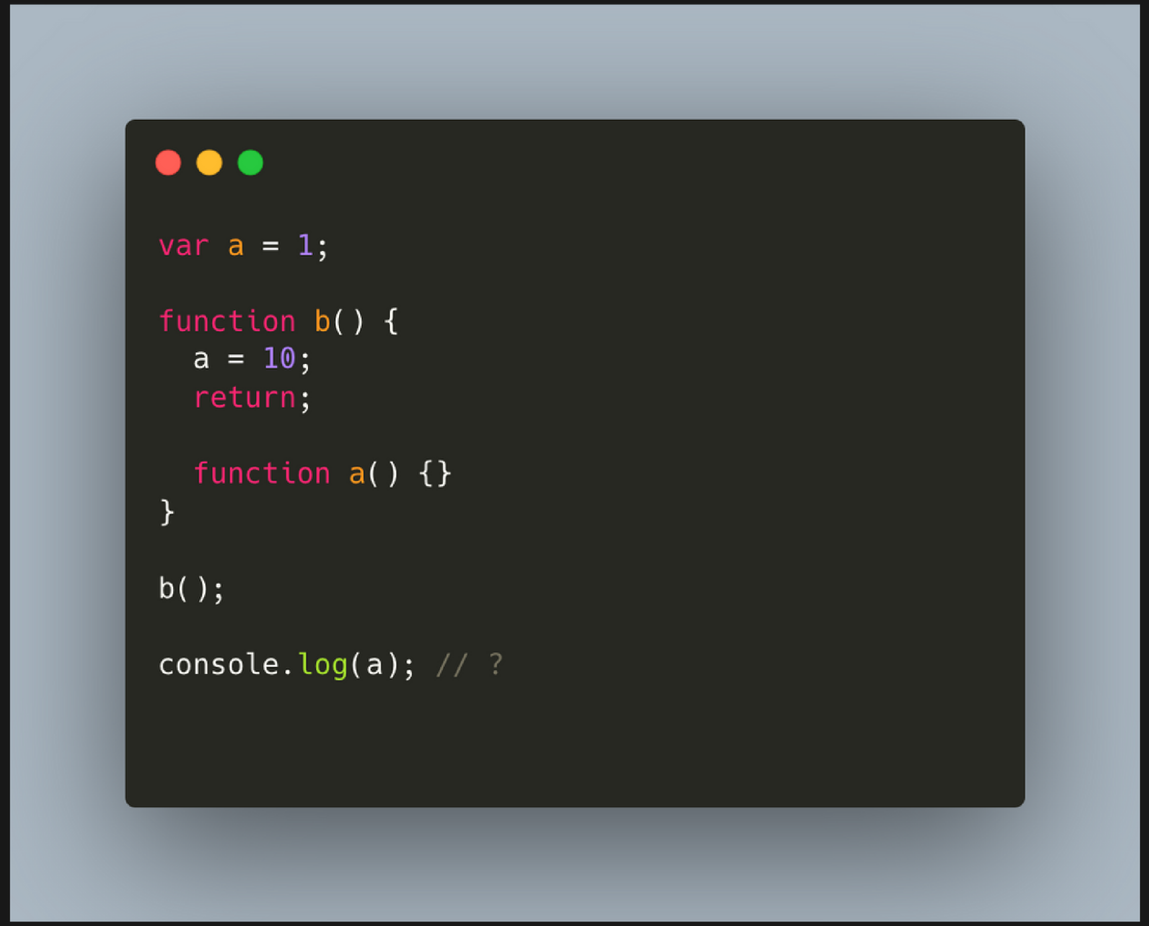
05. CREATE A PERSONAL BRAND

06. STAY UP-TO-DATE

06. COLLABORATE WITH OTHER JUNIORS,
CREATE A SMALL STARTUP PROJECT

CHALLENGE

11.1 CHALLENGE I

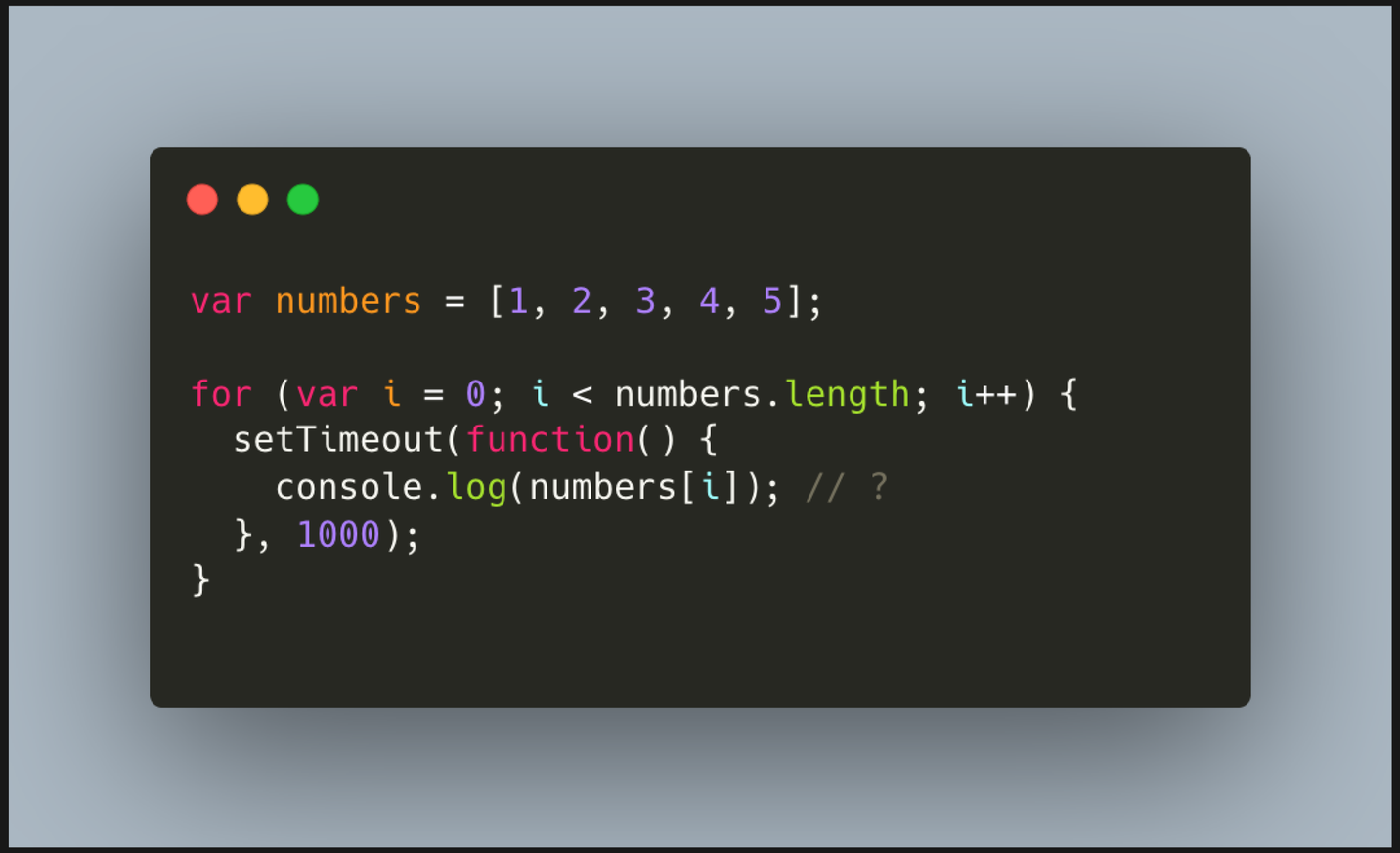


```
var a = 1;

function b() {
  a = 10;
  return;

  function a() {}
}

b();

console.log(a); // ?
```

11.1 CHALLENGE I - SOLVED

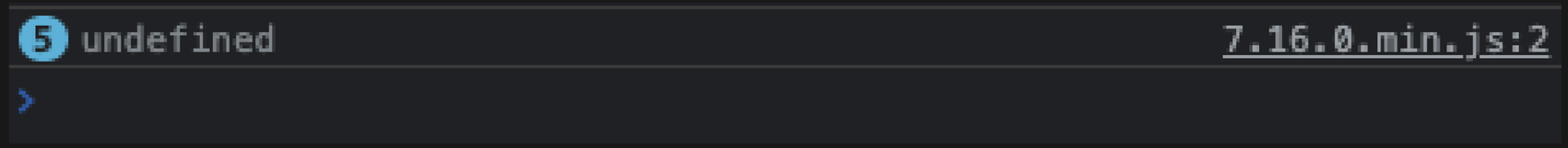
```
> var a = 1;
  function b() {
      a = 10;
      return;
      function a() {}
  }

  b();
  console.log(a);
  1                                          7.16.0.min.js:2
```

11.1 CHALLENGE II

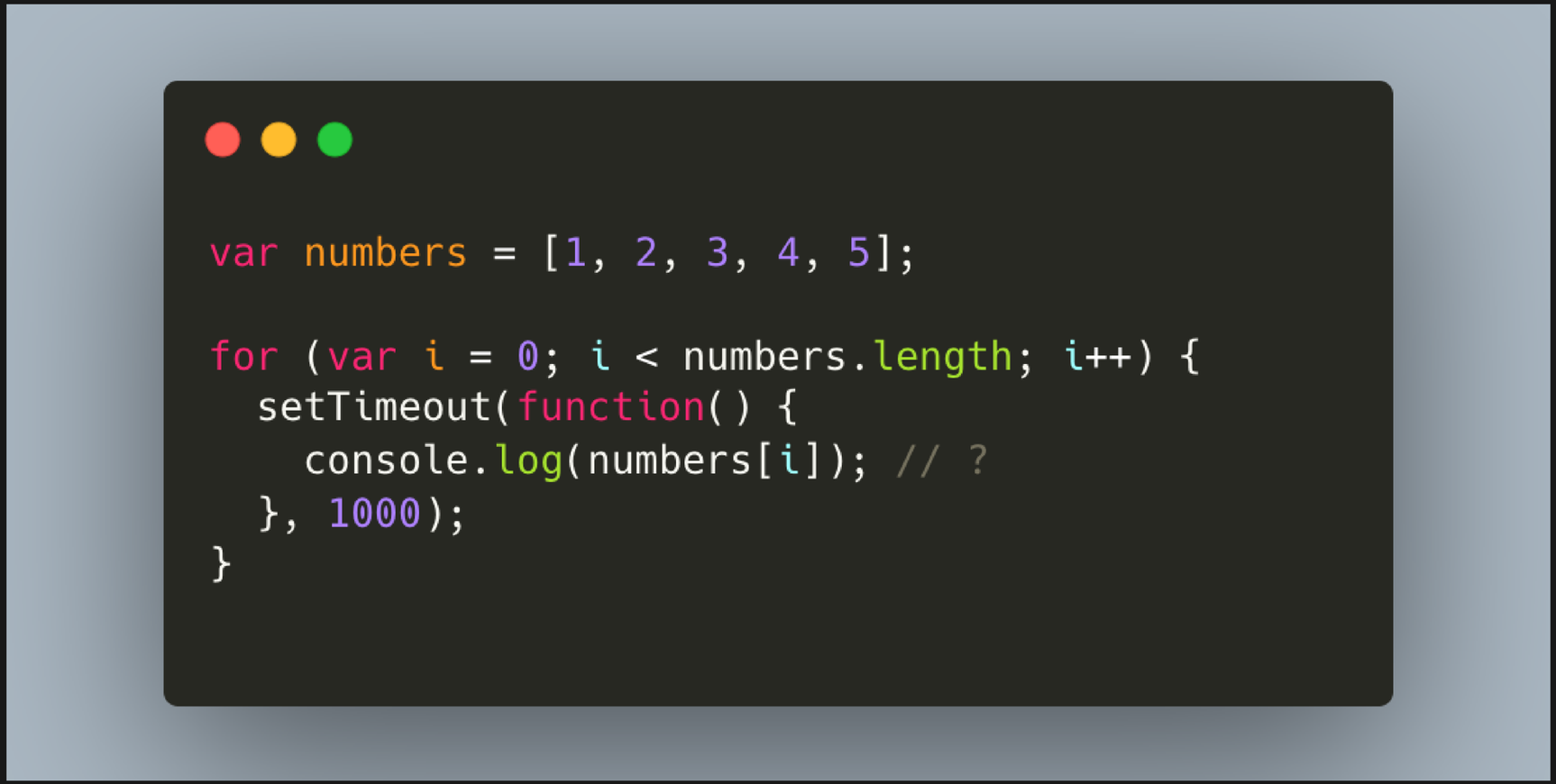


```
var numbers = [1, 2, 3, 4, 5];

for (var i = 0; i < numbers.length; i++) {
  setTimeout(function() {
    console.log(numbers[i]); // ?
  }, 1000);
}
```

11.1 CHALLENGE II - SOLVED



11.1 CHALLENGE III (FIX)



```
var numbers = [1, 2, 3, 4, 5];

for (var i = 0; i < numbers.length; i++) {
  setTimeout(function() {
    console.log(numbers[i]); // ?
  }, 1000);
}
```

11.1 CHALLENGE III (FIX) - SOLVED

```
var numbers = [1, 2, 3, 4, 5];

for(let i = 0; i < numbers.length; i++) {
    setTimeout(function() {
        console.log(numbers[i])
    }, 1000)
}
```

11.1 CHALLENGE III (FIX) - SOLVED

```
var numbers = [1, 2, 3, 4, 5];

for (var i = 0; i < numbers.length; i++) {
  (function(){
      var j = i;
    setTimeout(function() {
    console.log(numbers[j]);
  }, 1000)}
  )()
}
```

You can do it !

Thank you !