

Extracted from:

JavaScript Brain Teasers

Exercise Your Mind

This PDF file contains pages extracted from *JavaScript Brain Teasers*, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit <http://www.pragprog.com>.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printed versions; the content is otherwise identical.

Copyright © 2023 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The Pragmatic Bookshelf

Dallas, Texas

The
Pragmatic
Programmers

JavaScript Brain Teasers

Exercise Your Mind



Faraz K. Kelhini
edited by Margaret Eldridge

JavaScript Brain Teasers

Exercise Your Mind

Faraz Kelhini

The Pragmatic Bookshelf

Dallas, Texas



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

For our complete catalog of hands-on, practical, and Pragmatic content for software developers, please visit <https://pragprog.com>.

For sales, volume licensing, and support, please contact support@pragprog.com.

For international rights, please contact rights@pragprog.com.

Copyright © 2023 The Pragmatic Programmers, LLC.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

ISBN-13: 979-8-88865-050-9

Encoded using the finest acid-free high-entropy binary digits.

Book version: B1.0—October 26, 2023

Puzzle 1

Your Code Deserves a Lift

`your_code_deserves_a_lift/your_code_deserves_a_lift.js`

```
let temp = 25;

function displayTemperature() {
  console.log(`Current temperature: ${temp} °C`);
}

function forecastTemperature() {
  console.log(`Expected temperature: ${temp} °C`);
  var temp = 28;
}

displayTemperature();
forecastTemperature();
```

Guess the Output



Try to guess what the output is before moving to the next page.

You might have expected the output to be:

```
Current temperature: 25 °C
Expected temperature: 25 °C
```

But this code will actually output:

```
Current temperature: 25 °C
Expected temperature: undefined °C
```

Discussion

In JavaScript, all declarations are subject to *hoisting*. This includes `var`, `let`, `const`, `function`, `function*`, and `class` declarations. Hoisting involves the automatic relocation of these declarations to the scope's beginning. But, their initialization is deferred until the execution flow hits the line where hoisting occurred.

This process occurs before the code is executed, and it helps JavaScript handle references to variables and functions. Consider the following code:

```
// Create a variable
var a = 10;
```

This code is processed by JavaScript in the following manner:

```
var a;
// Create a variable
a = 10;
```

When you declare a variable using the `var` keyword, the declaration is moved (or “hoisted”) to the top of the containing function or global scope. However, the assignment (initialization) of the variable remains in its original place. This means that you can reference a `var` before it's declared in your code without causing an error. However, the value will be `undefined` until you actually assign a value to it.

This behavior can lead to unexpected outcomes when working with functions. Any variable that is declared within a function will be hoisted to the top. Consequently, if there exists a variable with an identical name in the global scope, it will become concealed within the function. For example:

```
var a = 10;

function fn() {
  console.log(a); // → undefined
  var a = 20;
  console.log(a); // → 20
}
```

```
fn();
```

So, when looking at the first line of this function, you might be thinking it'll log 10 and 20. But JavaScript actually processes the code differently:

```
var a = 10;
function fn() {
  var a;
  console.log(a); // → undefined
  a = 20;
  console.log(a); // → 20
}
fn();
```

To make sure things go smoothly and we don't run into any surprises, ES2015 came up with a solution: the `let` keyword for declaring variables. With `let`, variables still get hoisted, but now they're like reminders. They'll give us a heads-up if we try to use a variable that hasn't been declared, preventing any unexpected hiccups:

```
function fn() {
  console.log(a); // → undefined
  console.log(b); // → ReferenceError: b is not defined

  var a = 20;
  let b = 20

  console.log(a);
}
fn();
```

So, when you try to get hold of a variable before it's declared, with `var`, you'll get `undefined` as a response. On the other hand, if you attempt the same thing with `let`, you'll get an error message. As with `let`, if you try to use a `const` or `class` before declaring them, you'll get a `ReferenceError` thrown your way.

Temporal dead zone (TDZ) is a term dubbed by the JavaScript community to describe why it's not possible to access `let` and `const` before they are declared. `let` and `const` are both hoisted similar to `var` and function declarations, but there is a time span between entering the scope and being declared in which they cannot be accessed. This period is the temporal dead zone.

Any attempt to access variables while they are in TDZ causes a `ReferenceError`. Once execution reaches the declaration, the variable in TDZ is removed and they are allowed to be accessed. The TDZ exists to help us find bugs in our code. Trying to access a variable before declaring it is rarely intentional.

The Lifespan of Variables



The lifespan of variables within a function spans the duration of the function's execution. Once a function concludes its execution, all local variables within it are automatically cleared. In contrast, global variables remain in memory until the program/webpage is closed.

It's important to remember that TDZ applies only to the code block in which the variable is declared. Attempting to access a variable outside its scope does not throw an error because the variable is not in TDZ:

```
console.log(a);    // → undefined
{
  console.log(a);  // → ReferenceError
  let a = 10;
}
var a = 20;
```

Function Hoisting

Similar to variables, function declarations are hoisted to the top of their containing scope. But they are hoisted entirely, including both the name and the function body. This means you can call a function declared using function before the actual declaration in the code:

```
fn1();    // → Hello!
function fn1() {
  console.log("Hello!");
}
```

Function expressions, on the other hand, are not hoisted in the same way. Only the variable declaration is hoisted, and the function assignment occurs at the point where the variable is declared:

```
fn2();    // → TypeError: fn2 is not a function
var fn2 = function() {
  console.log("Hello!");
};
```

In this case, the variable `fn2` is initially assigned `undefined`, so you can't call it as a function until the assignment is made.

So, remember, it's best to declare variables at the top of the scope they belong to. That way, you'll steer clear of any confusion down the road. Additionally, try to get into the habit of using `let` and `const` instead of `var` because they help

prevent potential problems like variable hoisting and accidental variable redeclaration.

Further Reading

JavaScript Hoisting

<https://developer.mozilla.org/en-US/docs/Glossary/Hoisting>

The var statement

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/var>

The let declaration

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let>

The const declaration

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const>