The JavaScript Oxymoron

Kishan Chandrashekhar

Tismo Technology Solutions (P) Ltd.

29 January 2024

ABSTRACT

This article intends to demystify the commonly misunderstood concept of constants in JavaScript. Many leading names in the industry have pondered, published blogs, and even went on to hold multiple seminars and conference talks about the grammatical inaccuracy of various terminologies in programming. A well known example of this very topic is the seemingly “known to all, understood by none” usage of the phrase “constant variable”. In simple terms, the phrase is an oxymoron. And like certain terminologies in programming, oxymorons are generally misunderstood too. Only in appreciating their literary significance and structural beauty, can one begin to understand the simplicity and triviality of “constant variable”.

OXYMORONS IN ENGLISH

Oxymoron is a figure of speech where seemingly opposite terms show up next to each other. It is derived from two Greek words: oxys meaning ‘sharp’ and moronos meaning ‘dull’. As expected, the word itself is an oxymoron.

OXYMORONS IN JAVASCRIPT

The const keyword in JavaScript is used to declare a constant. Constants are often thought of as variables that cannot change:

```
> const hello = 3;
> hello = 6; // Uncaught TypeError: Assignment to constant variable.

> console.log(hello); // -> 3
```

The constant variable ‘hello’ is immutable. Despite described as a variable, it cannot be changed.
But on the other hand, consider the example below:

```
> const person = {
     name: 'Andy'
  };
> person.name = 'Murray';

> console.log(person); // -> { name: 'Murray' }
```

Turns out, creating an object using const does not necessarily mean it’s a constant. To better understand this behaviour, one must first learn the difference between assignment and mutation. The better understanding one has of these two key terms, the more sense JavaScript will make.

SIGNIFICANCE OF LABELS

The following two examples are perfectly valid JavaScript programs:

```
// Program 1
> 5;

 // Program 2
> ['cats', 'dogs', 'spiders'];
```

In both the examples above, a number and an array are displayed. These are created and stored in memory when the code runs. Now once they are in memory, how are they accessed?
It is understood that the number and the array are stored somewhere in memory, but they are not of any use as they cannot be accessed. This is where labels come in handy. Variables allow the option to stick a label on the things that have been created, so that they can referenced at the time of need.

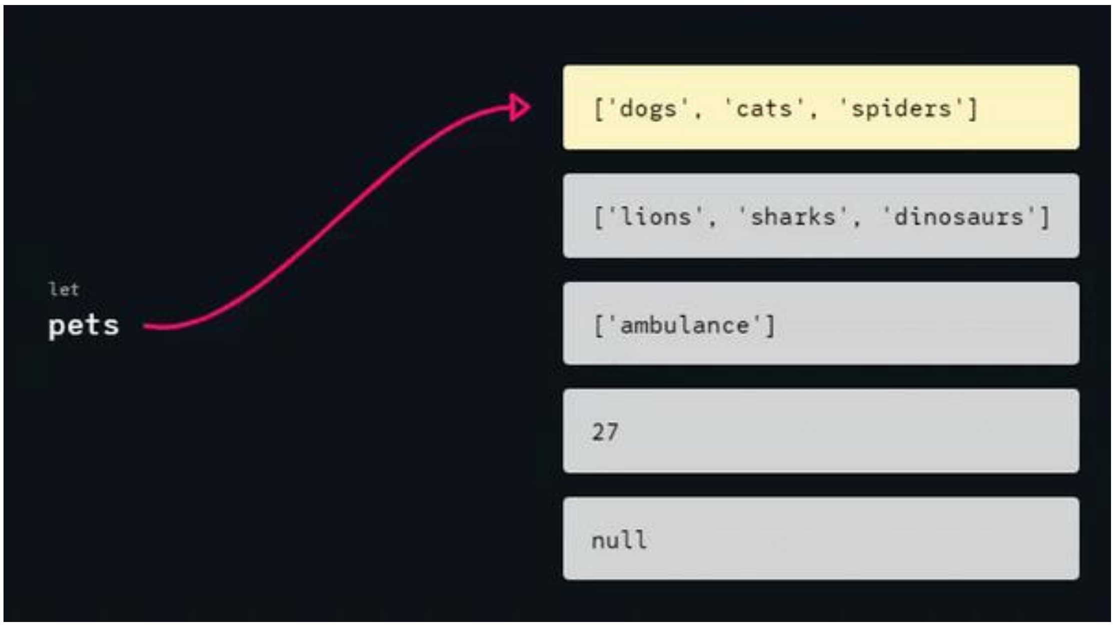
```
// Create it now
> const pets = ['cats', 'dogs', 'spiders'];

 // Reference it later
> console.log(pets); // -> ['cats', 'dogs', 'spiders'];
```

Since the general majority grew up learning English or other ‘sinistrodextral‘ languages, which are languages that read from left to right, it is then naturally assumed that code too gets executed from left to right.
So, the assumption one makes is that the `pets` variable is first created, like an empty slot, and then the array is assembled within that slot. This of course, is not the right mental representation. It is more accurate to say that the array gets created first, then the `pets` label is pointed at it.
Please note that the term “variable” can be a little bit confusing. Unfortunately, it is the established umbrella term in JavaScript to reference both `let` and `const`. Going forward, ‘variable’ will refer to any label attached to data.

LABEL REASSIGNMENT

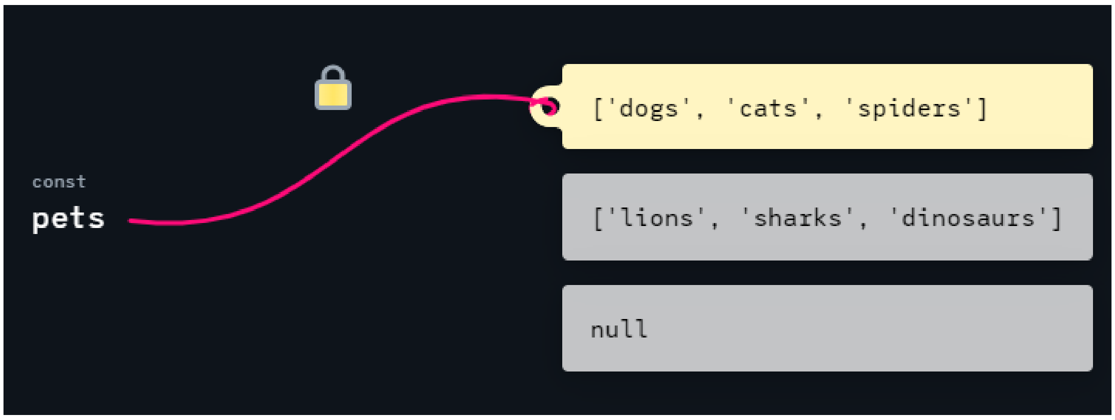
When one uses the `let` keyword to create a variable, they are able to change which “thing” that label refers to.
For instance, `pets` label can be pointed at a new value:

```
> let pets = ['dogs', 'cats', 'spiders'];
```



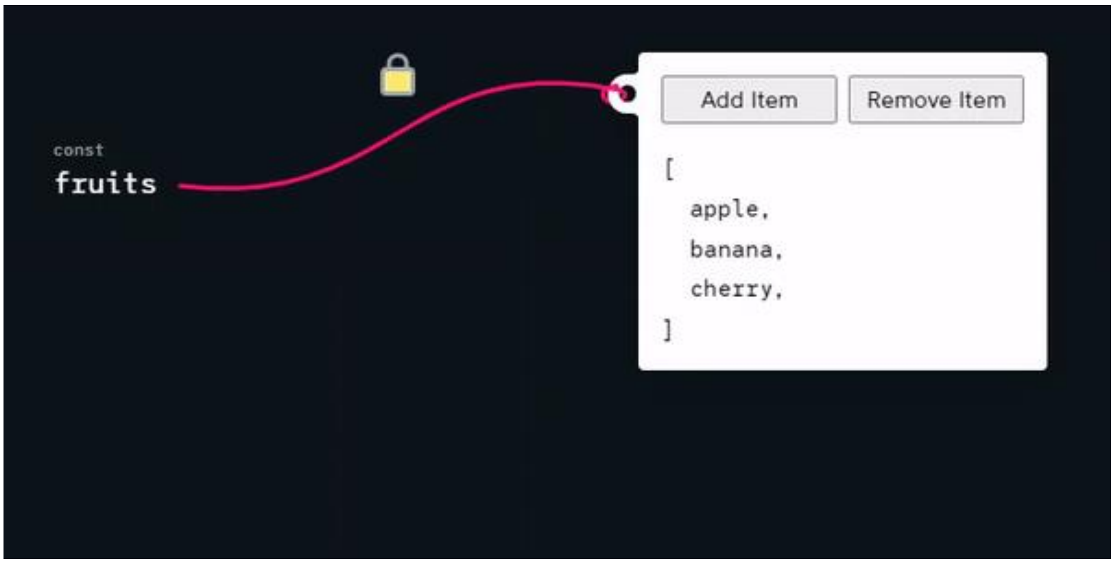
As shown in the above image, `pets` can be pointed to *[‘lions’, ‘sharks’, ‘dinosaurs’]* or *[‘ambulance’]* or *27* or *null* without throwing any errors.

This is known as re-assignment. It showcases that actually, the `pets` label should refer to an entirely different value. By doing so, the data is not modified, but the label is. The label gets detached from the original array and gets connected to a new one.
On the contrary, variables created with `const` cannot be reassigned.



That is the fundamental difference between `let` and `const`. When in need of an indestructible link between a variable name and a piece of data, use `const`.
Modification of the data itself is allowed, as long as the label remains intact. As shown below, addition or removal of items from an array can be performed without issues. The `fruits` variable is still connected to the same array:

```
// Points to the 'fruits' label at this array:
> const fruits = ['apple'];
```



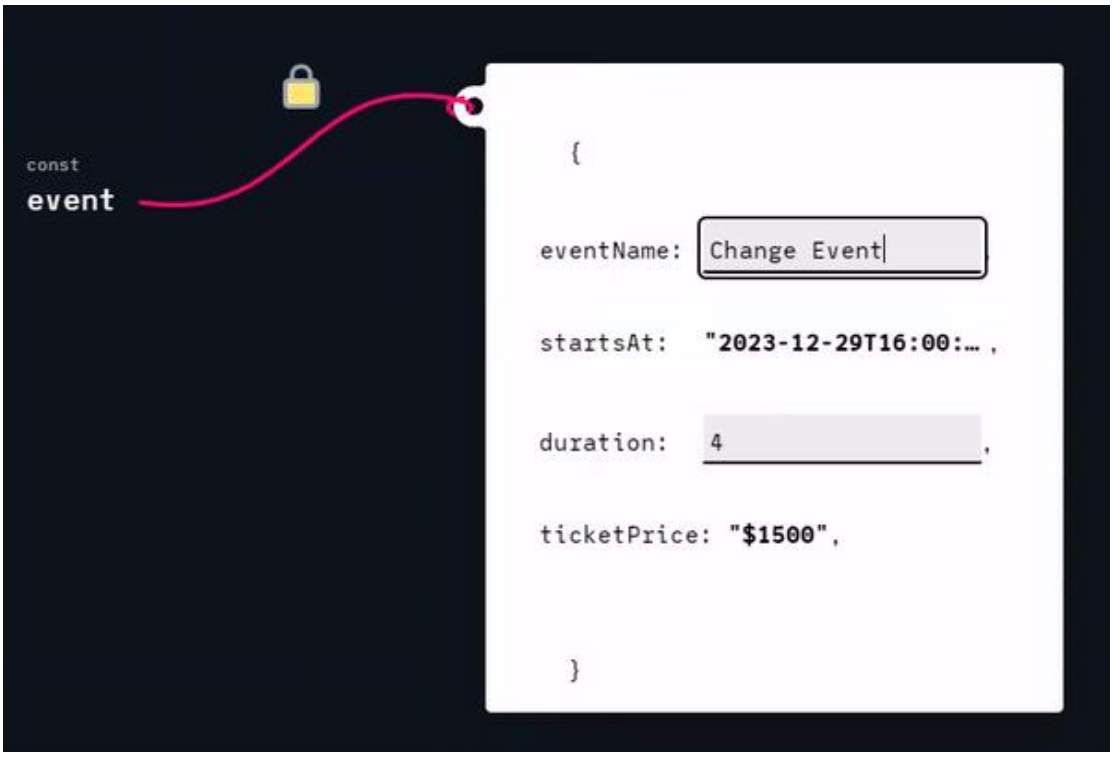
Via code, the following is valid:

```
// Points to the 'fruits' label at this array:
> const fruits = ['apple'];

 // Modify the array:
> fruits.push('banana');
> fruits.push('cherry');
> fruits.push('cement'); // Not meaningful, but valid!
```

This is known as mutation. The value of the array is edited by adding / removing items.
Let's look at another convincing example.

DOG SHOW OR TEA PARTY?

In the below example, an event slot has been booked, but it's not confirmed yet as to what the event is. So technically, one can keep any event in the slot and still have the slot be valid for the specified duration.



```
{
eventName: Change Event
startsAt: "2023-12-29T16:00:… ,
duration: 4 ,
ticketPrice: "$1500",
}
```

And this is the JS equivalent of what is happening above:

```
> const event = {
    eventName: 'Change event',
    startsAt: '2023-12-29T16:00:00Z',
    duration: 4,
    ticketPrice: '$1500'
  };

> event.eventName = 'Tea Party';
> event.duration = 6;
```

In layman terms, Re-assignment means pointing a variable name at a new thing while Mutation means editing the data within the thing. So, when a variable is created with `const`, the variable will never be re-assigned, but the same is not true when it comes to mutation. `const` is simply not designed to block mutation.

FROZEN IN TIME

There is a way to prevent mutations in JavaScript. *Object.freeze()* can be used to make the variable immutable.

```
// With arrays:
> const arr = Object.freeze([10, 20, 30]);

> arr.push(40);
> console.log(arr); // -> [10, 20, 30];

// With objects:
> const person = Object.freeze({ name: 'Veronica'});

> person.name = 'Betty';
> console.log(person); // -> {name: 'Veronica'};
```

CONCLUSION

In JavaScript (or any framework using JavaScript), default every variable to `const`. The compiler will indicate when there is an attempt by the user to edit a constant. In those scenarios, just change it from `const` to `let`.

It is recommended to default every variable created in JavaScript (or any framework using JavaScript) to `const`. On situations where an attempt is being made to edit a const variable,

REFERENCES

1. Use `const` and make your JavaScript code better : https://medium.com/dailyjs/use-const-and-make-your-javascript-code-better-aac4f3786ca1

2. [JavaScript] - What does const mean in JavaScript? : https://www.shecodes.io/athena/37774-what-does-const-mean-in-javascript#:~:text=In%20JavaScript%2C%20const%20is%20a,once%20it%20has%20been%20declared

3. 100 Awfully Good Examples of Oxymorons : https://www.thoughtco.com/awfully-good-examples-of-oxymorons-1691814