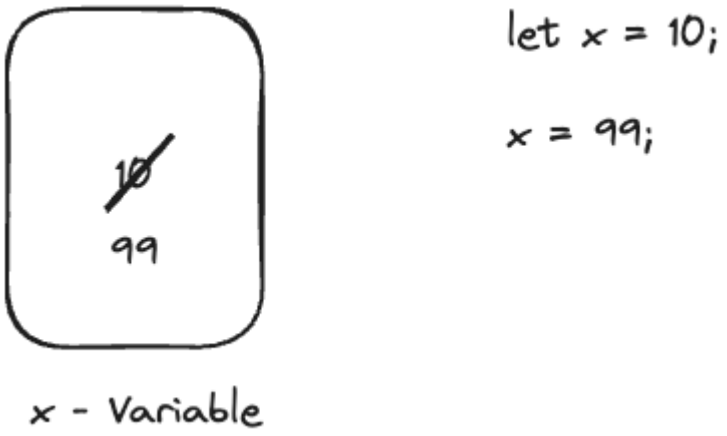
Mutability and Immutability in JS

In most of the programming languages, this concept of mutability and immutability exists.

First of all let's try to understand what is the meaning of the word `mutable`.



In the above code, we can see that we initialise a variable `x` with a value 10, and later we are able to update it to 99. This mechanism of being able to update / modify a variable's value is called as mutability. If we can modify a variables value then it is mutable.

That's where immutability also comes, when we cannot do the modifications.

Making immutable values with const

We can use `const` based initialisations for our variables to make them immutable. If we have a variable initialised by `const` then we will not be able to update / modify / reassign a value to the same variable.

```
const y = 100;
y = 99;
```

Then the above code throws the following error:

```
TypeError: Assignment to constant variable.
    at <anonymous>:1:3
```

One more interesting fact about `const` based initialisation is that, we cannot leave a const variable undefined. Like it's not possible to just declare the variable and leave it. You have to give some initialisation value be it a number, undefined, null etc
The following code throws an error:

```
const z;
```

This code throws an error saying:

```
SyntaxError: Missing initializer in const declaration
```

But we can do something like this:

```
const z = undefined;
```

The above code works now.

Why do we need immutability ?

There are a lot of places in programming where immutability becomes extremely important. Let's say we have a DB connection, there are very less cases where we want to update the configuration of this DB connection. Most of the time, once the connection object is setup we don't want to change it, this is a good example of immutability requirement.

Let's say you have some secret keys being used in your project, we don't want any piece of code to be able to manually change them once they are initialised, again a requirement of immutability.

Pros:

- Makes important configurations non-modifiable
- Less complex to manage
- Less memory needed

Cons:

- We need to always keep in mind what to make mutable and what to make immutable.
- Handling it in objects is tricky.

Why is it tricky to handle immutability in objects ?

If we have a `const` based initialisation for objects then it doesn't make the object fully immutable.
Even if we have a `const` based object, we will be able to very easily update value of a key and add more key value pairs, that means we can update the object.

```
const obj = {x: 10, y: 20};
obj.x = 99; // allowed to update value of a key
obj.z = 98; // addition of more key value pair allowed
```

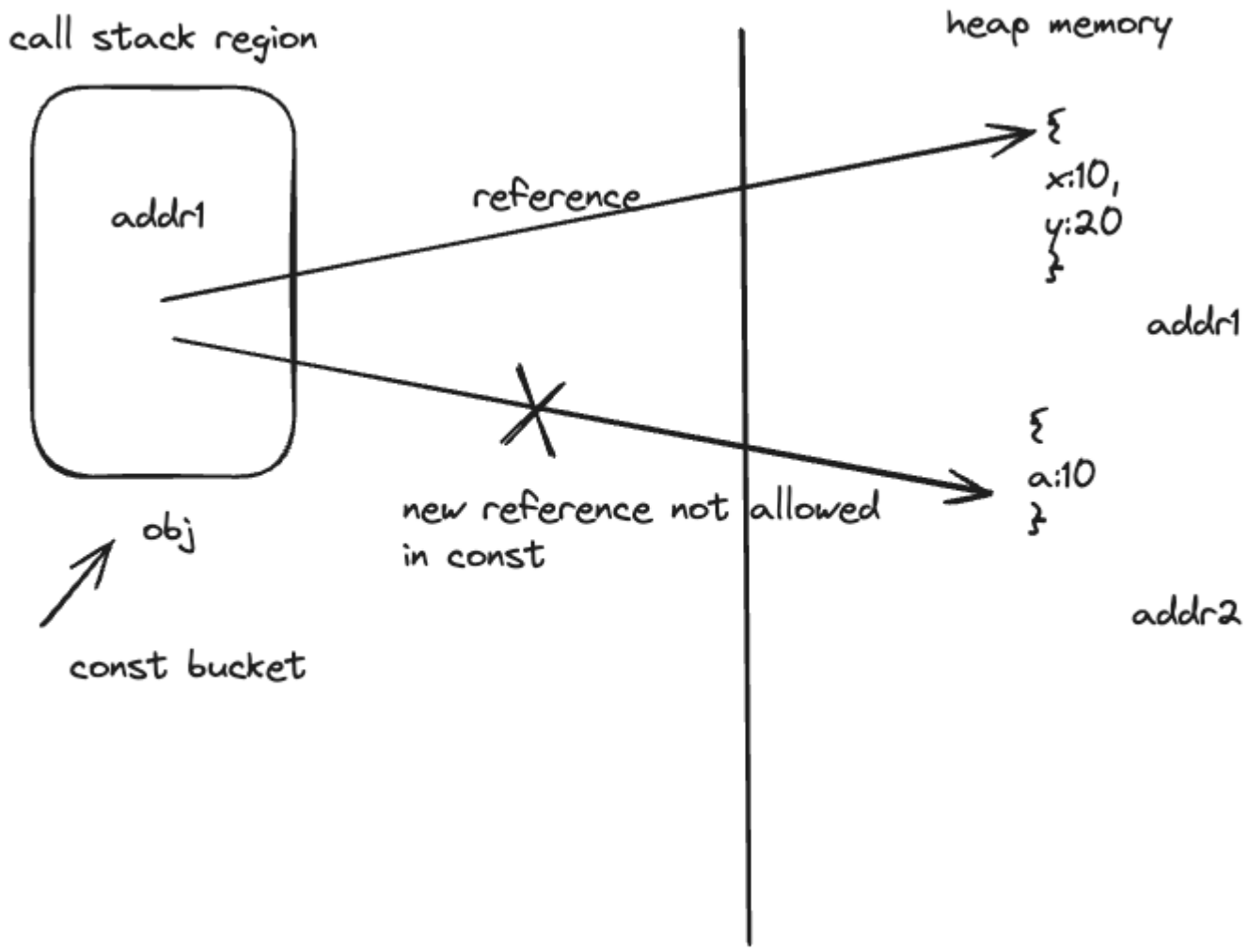
So what is `const` doing here ?

`const` mainly stops re-assignment of variables. It will stop when we will try to reassign an absolutely new object to our variable.

```
obj = {a : 10}; // this is not allowed
```

The above code throws the error:

```
TypeError: Assignment to constant variable.
```



Whenever we make an object, the variable bucket stays in the stack memory, but object is created in the heap memory of the process (heap is a big pool of unstructured memory) and the reference of the object is stored in the variable bucket (by storing the address).

So, if we try to reassign a new object, then we are changing the reference and that's not allowed by const.

So technically, with `const` objects are not fully immutable.

Can we make objects fully immutable ?

So there are two important methods, `Object.seal` and `Object.freeze` using which we can achieve immutability in objects.

Object.seal

This method takes one argument, i.e. the object we want to make immutable. This method will help us to make sure that we are not able to add new key value pairs, or delete an existing key

value pair.

But it will allow update of existing key value pairs.

Example

```
const product = {name: "Iphone 14 prp", price: 125000}
Object.seal(product);

product.company = "apple"; // new addition not allowed

console.log(product); // this will still print - {name: 'Iphone 14 prp',
price: 125000}

delete product.price; // deletion of key-value pair not allowed

console.log(product); // this will still print - {name: 'Iphone 14 prp',
price: 125000}

product.name = "Iphone 14 pro";
console.log(product); // this will update the key value pair - {name:
'Iphone 14 pro', price: 125000}
```

Object.freeze

This provides highest level of immutability. It creates a frozen object which means

- We cannot add new key value pairs
- We cannot remove existing key value pairs
- We cannot update existing key value pairs

```
const product = {name: "Iphone 14 prp", price: 125000}
Object.freeze(product);

product.company = "apple"; // new addition not allowed

console.log(product); // this will still print - {name: 'Iphone 14 prp',
price: 125000}

delete product.price; // deletion of key-value pair not allowed

console.log(product); // this will still print - {name: 'Iphone 14 prp',
price: 125000}
```

```
product.name = "Iphone 14 pro";
console.log(product); // this will still print - {name: 'Iphone 14 prp',
price: 125000}
```

Object.isFrozen and Object.isSealed

These methods will help us to check, if the objects are manually sealed and frozen or not ?

If an object is frozen then it will return true for both isFrozen and isSealed.

```
console.log(Object.isFrozen(product)); // true
console.log(Object.isSealed(product)); // true
```

If an object is sealed manually then it will only return true for isSealed and false for isFrozen.

```
const x = {a: 10};
Object.seal(x);
console.log(Object.isFrozen(product)); // false
console.log(Object.isSealed(product)); // true
```

If object is neither sealed nor frozen then it returns false for both.

Object.preventExtension

This method helps us to achieve 50% of what seal gives us.

- We cannot add new key value pairs
- We can remove existing key value pairs.
- Update is allowed.

```
const product = {name: "Iphone 14 prp", price: 125000}
Object.preventExtension(product);

product.company = "apple"; // new addition not allowed

console.log(product); // this will still print - {name: 'Iphone 14 prp',
price: 125000}

delete product.price; // deletion of key-value pair not allowed
```

```
console.log(product); // this will still print - {name: 'Iphone 14 prp'}

product.name = "Iphone 14 pro";
console.log(product); // this will still print - {name: 'Iphone 14 prp'}
```

Object.defineProperty

This function is very powerful as it gives us granular control on make any particular set of key value pairs (from all the available pairs) as writable or configurable.
Writable means, can we update the key value pair or not ?
Configurable means, can we update key value pair or not ?
This method takes first argument as object, second argument as the key which we want to work on and then third argument is a new object which has `configurable` and `writable` booleans.

```
Object.defineProperty(product, 'name', {configurable: false, writable:
false});
```

This statement will make the name key as non updatable and non deletable. But other keys like price can be updated and deleted as well.

```
Object.defineProperty(product, 'name', {configurable: false});
```

This statement will make the name key as non deletable but we can update it. But other keys like price can be updated and deleted as well.

```
Object.defineProperty(product, 'name', {writable: false});
```

This statement will make the name key as non updatable but we can delete it. But other keys like price can be updated and deleted as well.

Can we make our own Object.seal ?

Yes, we can combine `Object.preventExtensions` and `Object.defineProperty` to make our own custom seal method. We can go to each each key of the object and make it non deletable.

```
function customSeal(obj) {
        let keys = Object.keys(obj);
        for(let i = 0; i < keys.length; i++) {
                Object.defineProperty(obj, keys[i], {configurable: false});
```

```
// this will stop deletion of key value pair
        }
        Object.precentExtensions(obj); // this will stop addition of new key
value pairs

}
```

Can we make our own Object.freeze ?

Yes, we can combine `Object.preventExtensions` and `Object.defineProperty` to make our own custom seal method. We can go to each each key of the object and make it non deletable and non updatable.

```
function customFreeze(obj) {
        let keys = Object.keys(obj);
        for(let i = 0; i < keys.length; i++) {
                Object.defineProperty(obj, keys[i], {configurable: false,
writable: false}); // this will stop deletion and updation of key value pair
        }
        Object.precentExtensions(obj); // this will stop addition of new key
value pairs

}
```