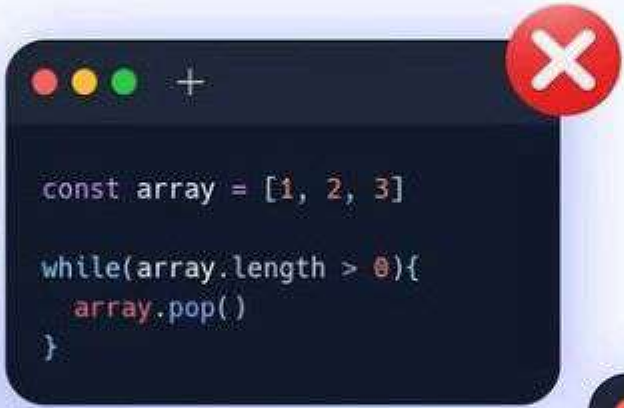
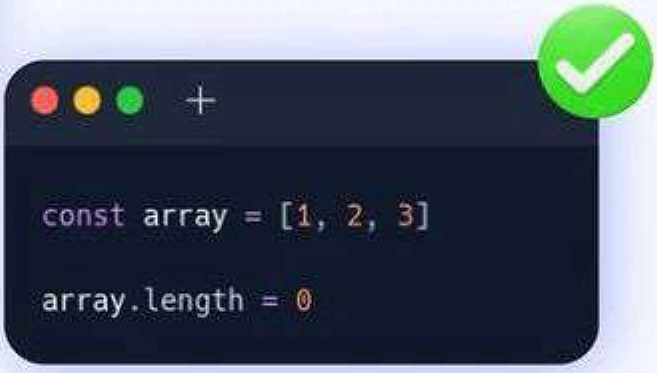
JavaScript Pro Tips You Need To Know

Empty an array by changing the length

Instead of **looping** and **popping** all the values, there is a handy trick that we can use in JavaScript to empty an array by manipulating the length. Simply making the **length** of array **0** will result it being empty.



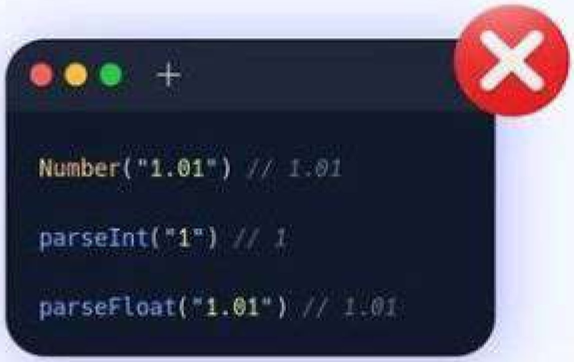
```
const array = [1, 2, 3]

while(array.length > 0){
  array.pop()
}
```



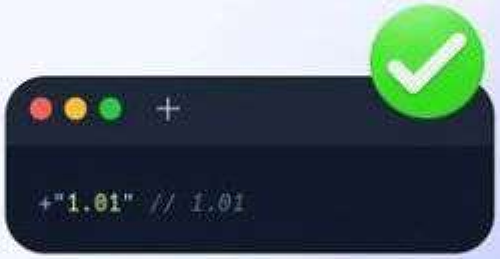
```
const array = [1, 2, 3]

array.length = 0
```

Use '+' operator to convert string to number

Usually to convert **string** to **number** we use wither **Number** class or **parseInt** or **parseFloat**. But you can actually convert any valid string to number by appending just a '**+**' operator at the beginning.



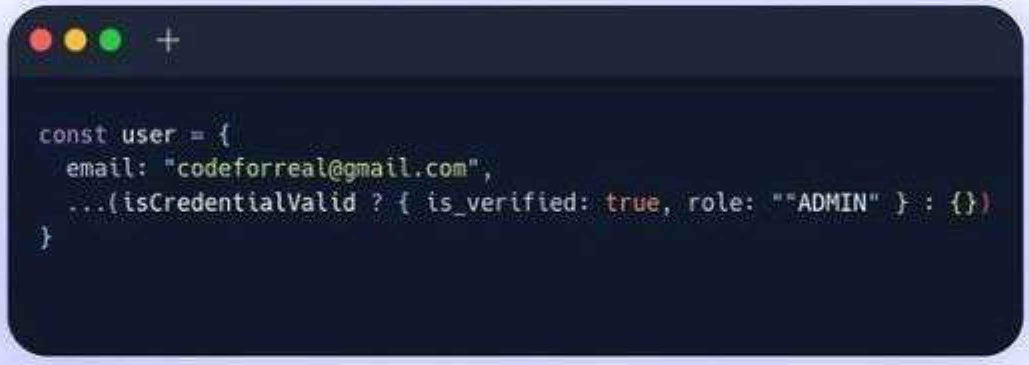
```
Number("1.01") // 1.01

parseInt("1") // 1

parseFloat("1.01") // 1.01
```



```
+"1.01" // 1.01
```

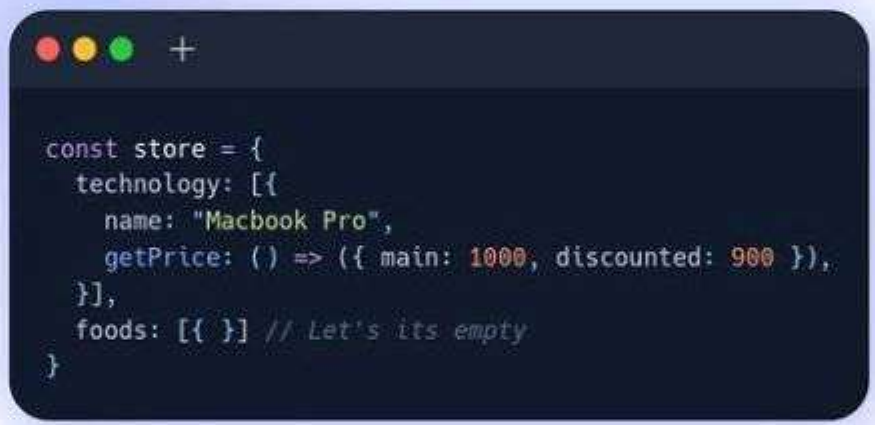
Conditionally add properties to an object

In JavaScript you can **conditionally** add properties to an object by using spread operator. **But why?** So while passing objects to something like **ORM**(**Object Relational Mapper**), we might not want to pass **undefined** or null **unintentionally**. So this is very useful in those cases where we do not want to assign a property when certain condition is met.



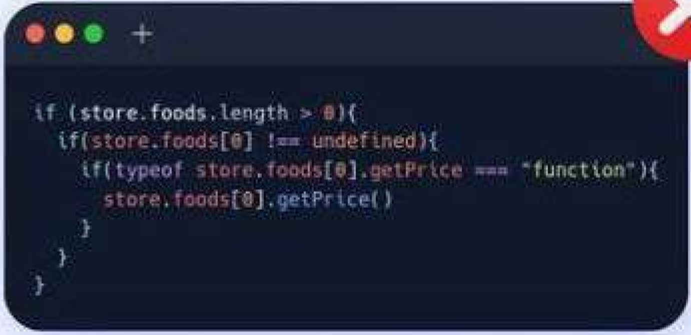
```
const user = {
  email: "codeforreal@gmail.com",
  ...(isCredentialValid ? { is_verified: true, role: ""ADMIN" } : {})
}
```

Optional Chaining (?)

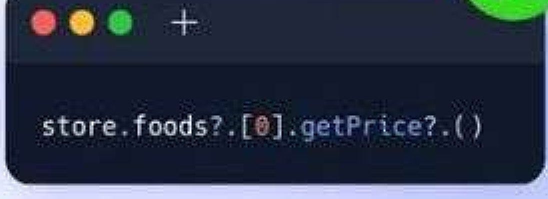


```
const store = {
  technology: [{
    name: "Macbook Pro",
    getPrice: () => ({ main: 1000, discounted: 900 }),
  }],
  foods: [{ }] // Let's its empty
}
```

Let's imagine we have this object



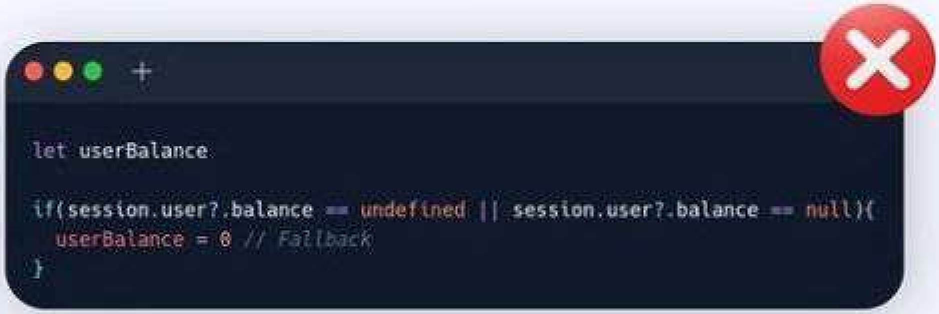
```
if (store.foods.length > 0){
  if(store.foods[0] !== undefined){
    if(typeof store.foods[0].getPrice === "function"){
      store.foods[0].getPrice()
    }
  }
}
```



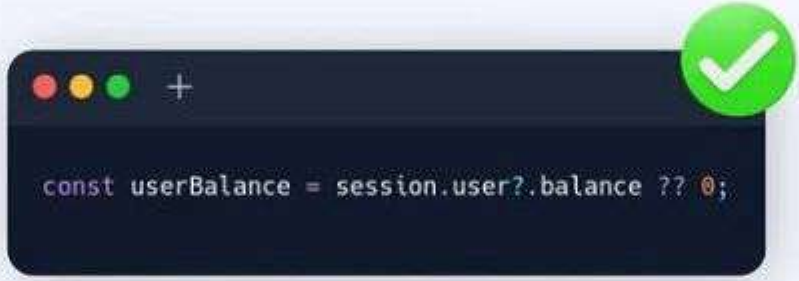
```
store.foods?.[0].getPrice?.()
```

Nullish Coalescing (??)

Let's say if you ever get into a situation where you need to assign a different value if it is either null or undefined only, then you can use nullish coalescing operator(**??**).



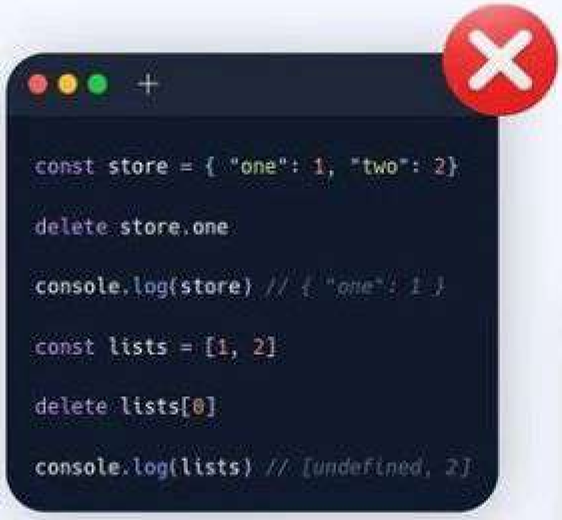
```
let userBalance

if(session.user?.balance == undefined || session.user?.balance == null){
  userBalance = 0 // Fallback
}
```



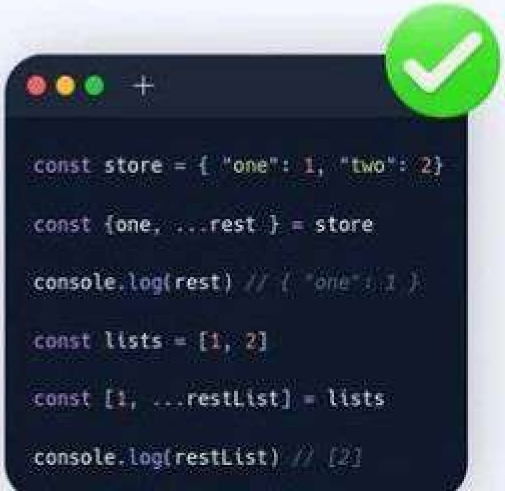
```
const userBalance = session.user?.balance ?? 0;
```

Do not use delete to remove entries from objects or arrays

delete is found to have some performance drawback since it needs to recreate the object. It also makes the entry **undefined** in an array. So a better approach is to use rest(**...**) opeartor to create a new reference instead of touching original.



```
const store = { "one": 1, "two": 2}

delete store.one

console.log(store) // { "one": 1 }

const lists = [1, 2]

delete lists[0]

console.log(lists) // [undefined, 2]
```



```
const store = { "one": 1, "two": 2}

const {one, ...rest } = store

console.log(rest) // { "one": 1 }

const lists = [1, 2]

const [1, ...restList] = lists

console.log(restList) // [2]
```