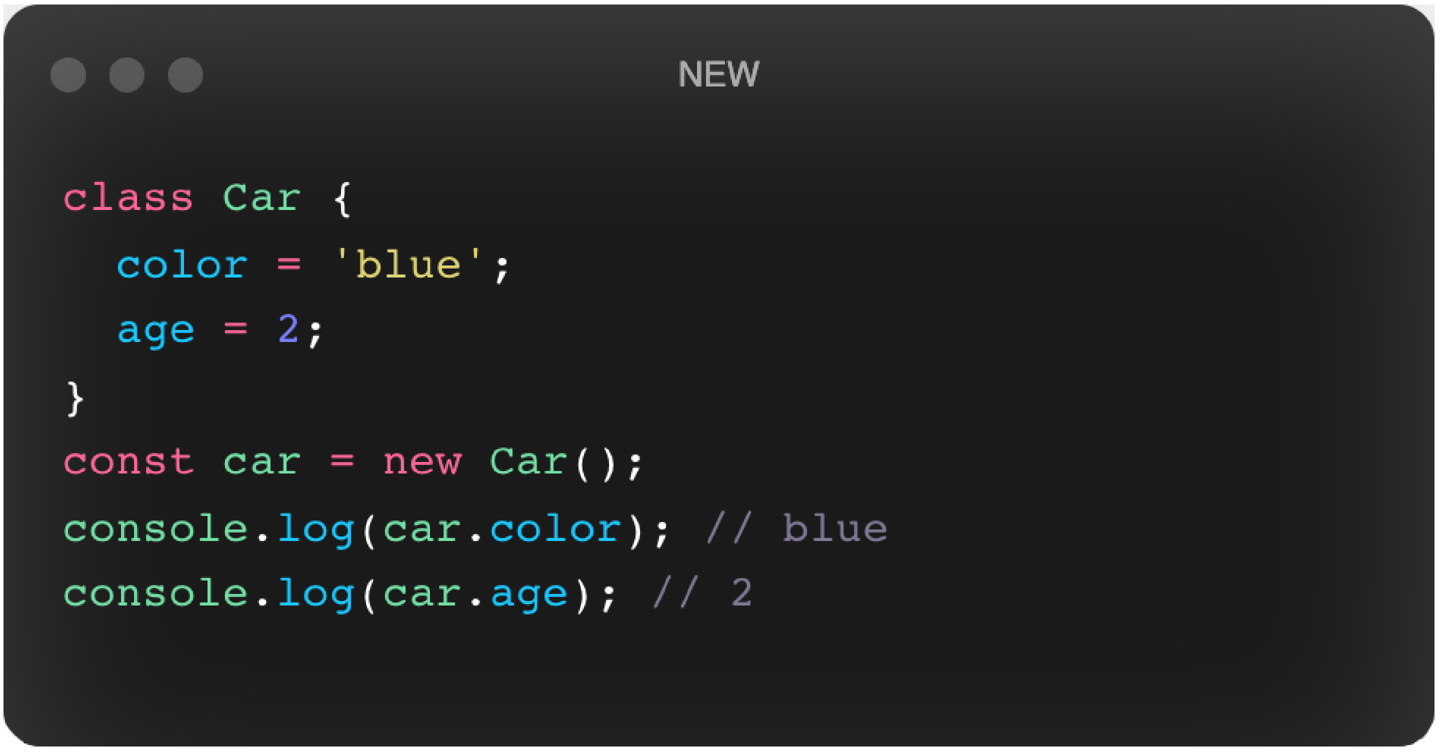
6 JavaScript ES13 Features

swipe

Class Field Declaration

Before ES13, class fields could only be declared in the constructor. Unlike in many other languages, we could not declare or define them in the outermost scope of the class.

ES13 removes this limitation. Now we can write code like this:


NEW

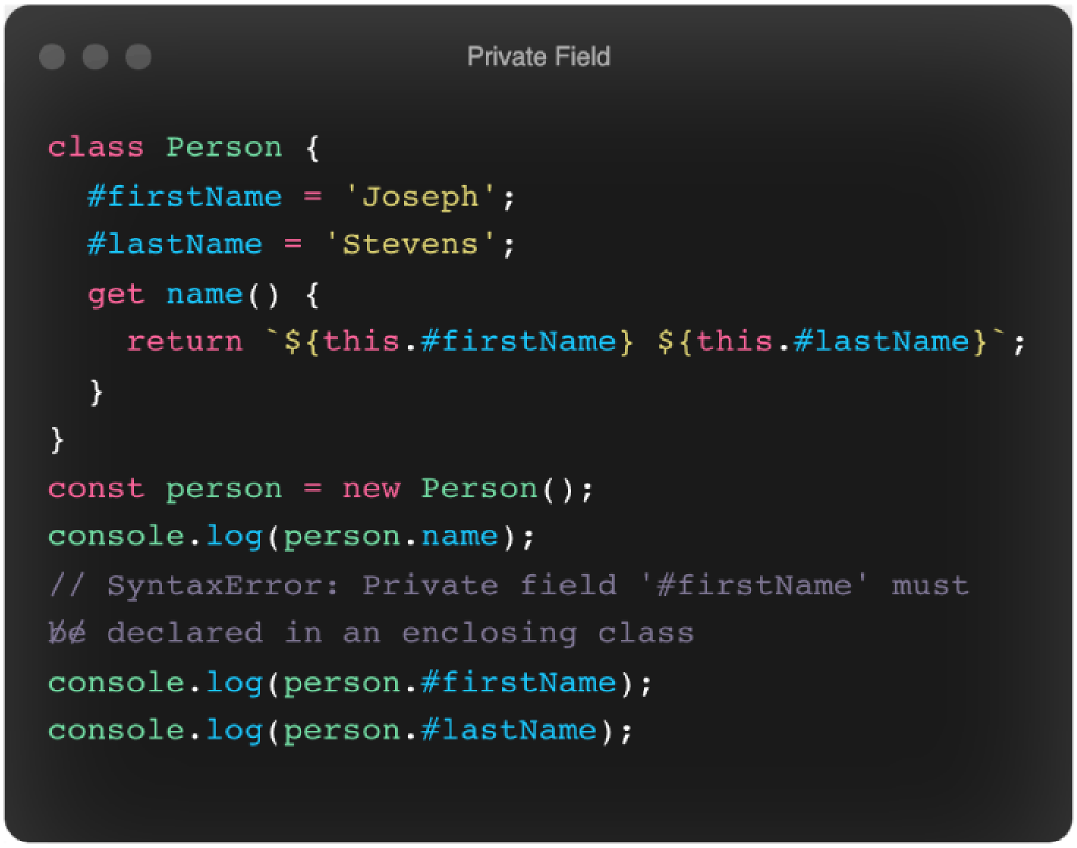
```
class Car {
  color = 'blue';
  age = 2;
}
const car = new Car();
console.log(car.color); // blue
console.log(car.age); // 2
```

swipe

Private Methods & Fields

Previously, it was not possible to declare private members in a class. A member was traditionally prefixed with an underscore **(_)** to indicate that it was meant to be private, but it could still be accessed and modified from outside the class.

With ES13, we can now add private fields and members to a class, by prefixing it with a hashtag **(#)**. Trying to access them from outside the class will cause an error:



Private Field

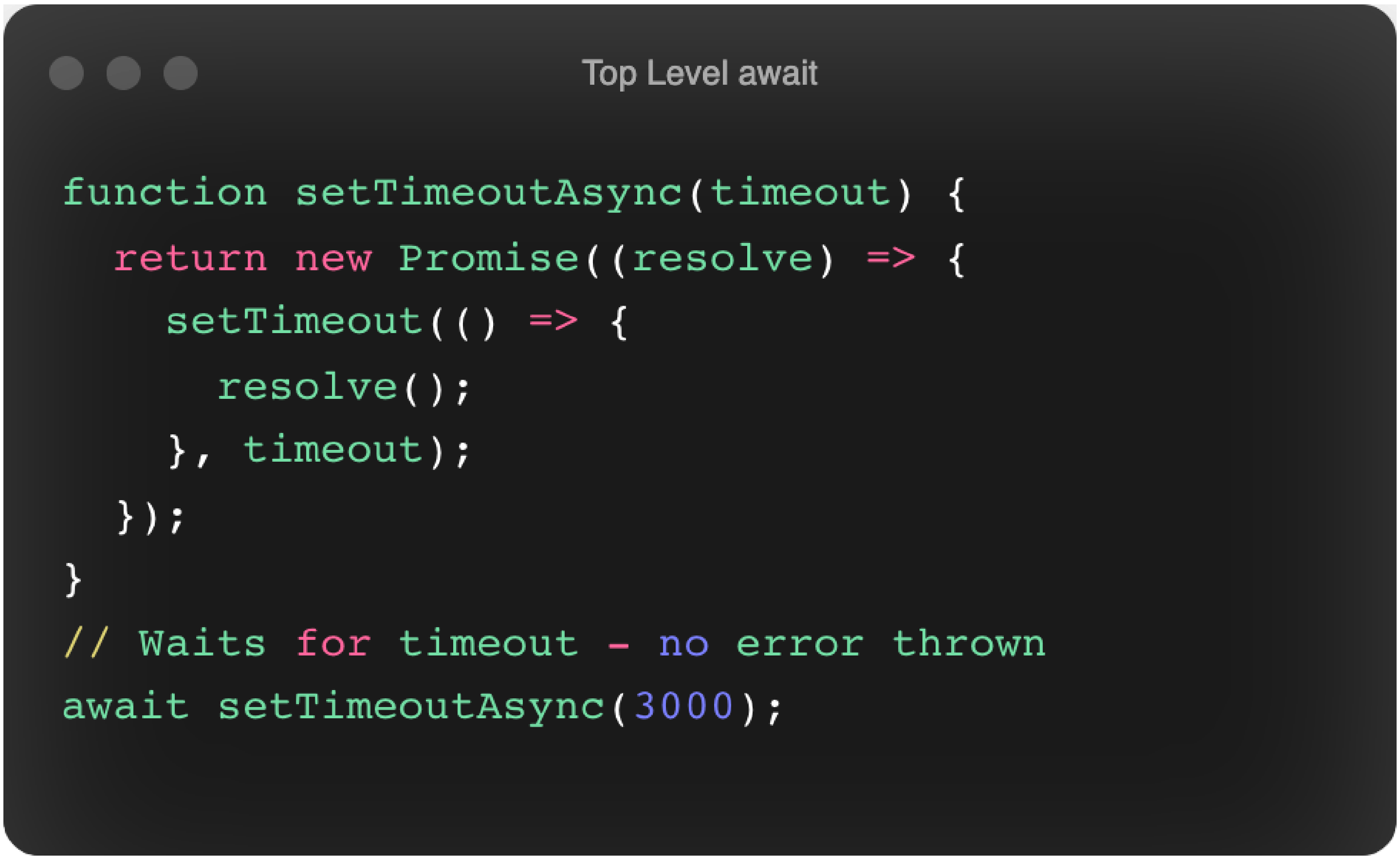
```
class Person {
  #firstName = 'Joseph';
  #lastName = 'Stevens';
  get name() {
    return `${this.#firstName} ${this.#lastName}`;
  }
}
const person = new Person();
console.log(person.name);
// SyntaxError: Private field '#firstName' must
be declared in an enclosing class
console.log(person.#firstName);
console.log(person.#lastName);
```

swipe

await at the Top Level

In JavaScript, the await operator is used to pause execution until a Promise is settled (fulfilled or rejected).

Previously, we could only use this operator in an async function - a function declared with the async keyword. We could not do so in the global scope. With ES13, now we can:



```javascript
function setTimeoutAsync(timeout) {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve();
    }, timeout);
  });
}
// Waits for timeout - no error thrown
await setTimeoutAsync(3000);
```

swipe

at() Method for Indexing

We typically use square brackets ([]) in JavaScript to access the Nth element of an array, which is usually a simple process. We just access the N - 1 property of the array.

The new at() method lets us do this more concisely and expressively. To access the Nth element from the end of the array, we simply pass a negative value of -N to at().

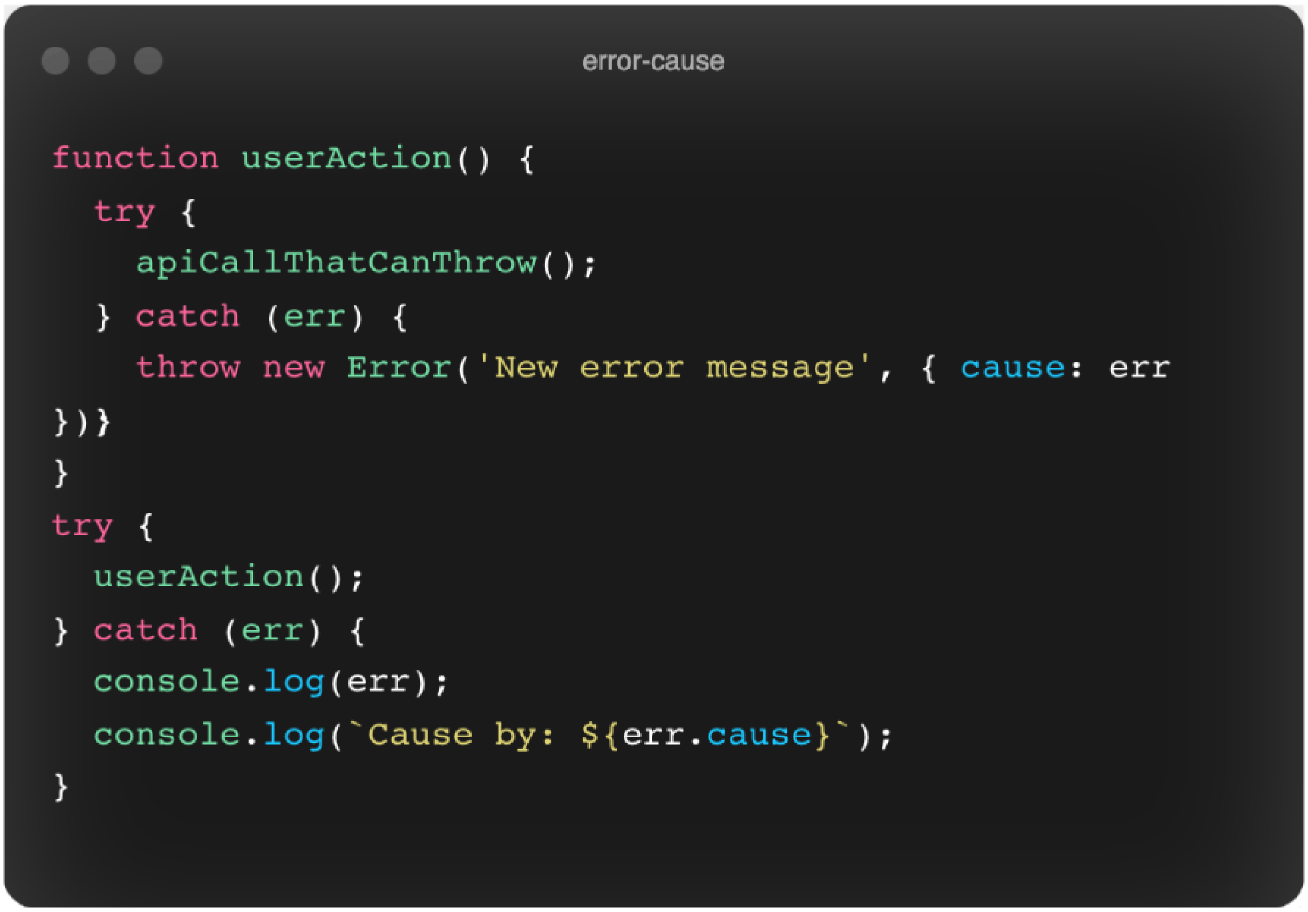
at()

```
const arr = ['a', 'b', 'c', 'd'];
// 1st element from the end
console.log(arr.at(-1)); // d
// 2nd element from the end
console.log(arr.at(-2)); // c
```

swipe

Error Cause

Error objects now have a **cause** property for specifying the original error that caused the error about to be thrown. This helps to add additional contextual information to the error and assist the diagnosis of unexpected behavior. We can specify the cause of an error by setting a cause property on an object passed as the second argument to the **Error()** constructor.



error-cause

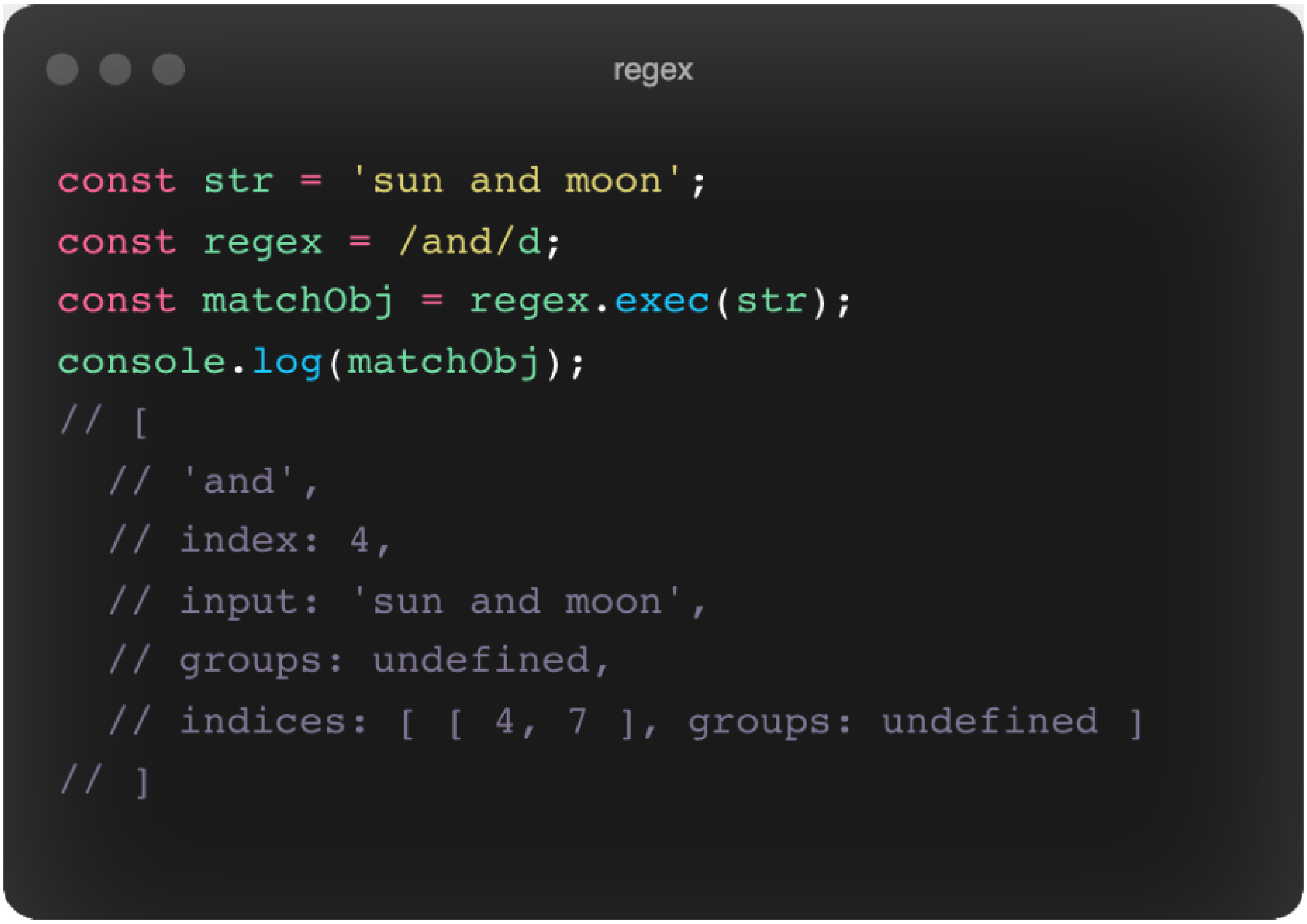
```
function userAction() {
  try {
    apiCallThatCanThrow();
  } catch (err) {
    throw new Error('New error message', { cause: err
})}
}
try {
  userAction();
} catch (err) {
  console.log(err);
  console.log(`Cause by: ${err.cause}`);
}
```

swipe

RegExp Match Indices

This new feature allows us to specify that we want the get both the starting and ending indices of the matches of a **RegExp** object in a given string.

Previously, we could only get the starting index of a regex match in a string. We can now specify a **d** regex flag to get the two indices where the match starts and ends.



regex

```
const str = 'sun and moon';
const regex = /and/d;
const matchObj = regex.exec(str);
console.log(matchObj);
// [
  // 'and',
  // index: 4,
  // input: 'sun and moon',
  // groups: undefined,
  // indices: [ [ 4, 7 ], groups: undefined ]
// ]
```

swipe

@the.tech.lead

Follow for more tips like this

Let us know what you think in the comments!

Give us some love

Save for later