Save it

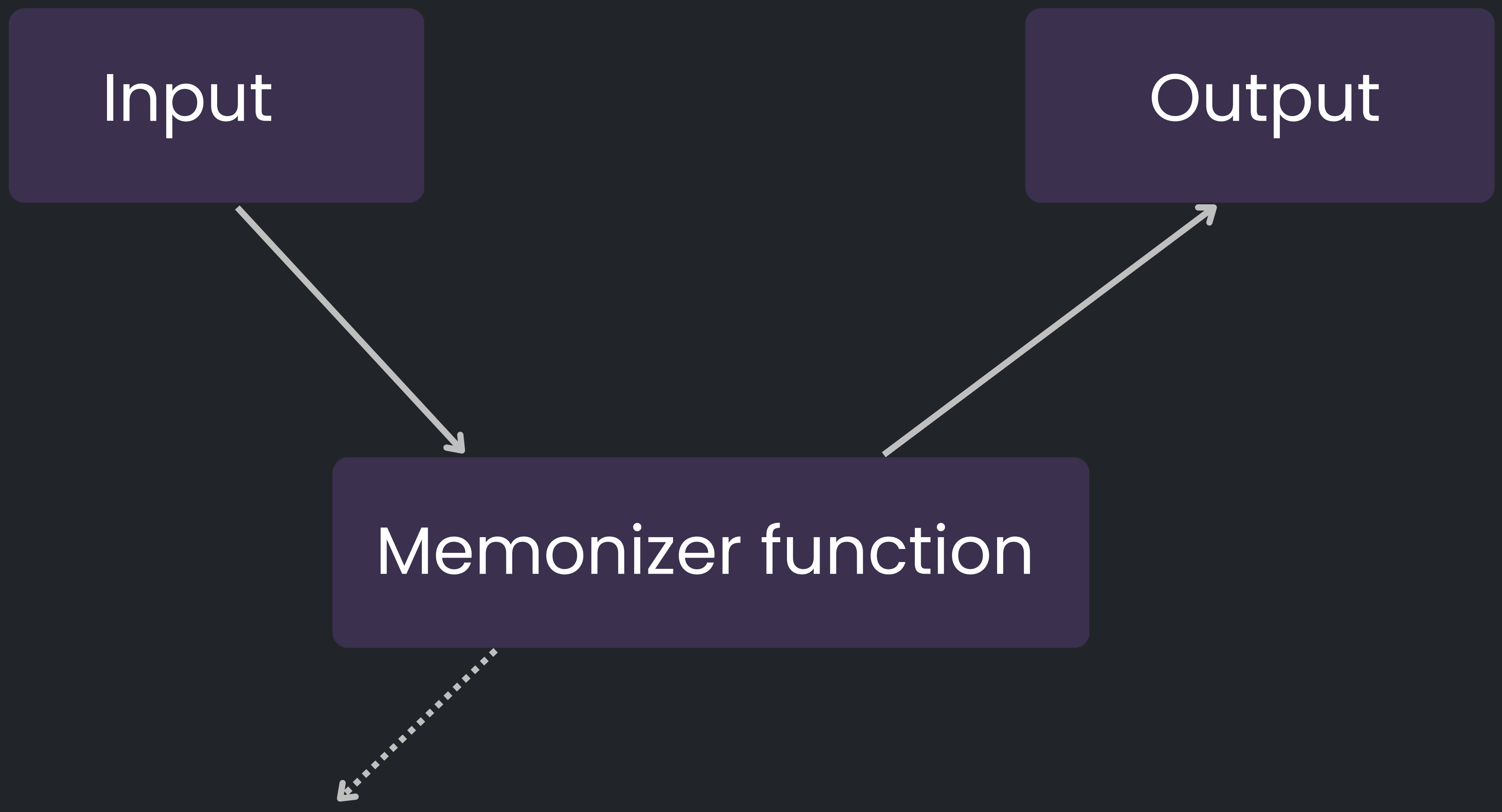
Like it

JS

Cache function result using **Memoization**

@coder_aishya

Flow using memoization



Check in cache object,
whether same request is passed ?

if present in cache return the cache value stored in cache else compute the result

What is Memoization ?

Memoization is an optimization technique where expensive function calls are cached(stored) such that the result can be immediately returned the next time the function is called with the same arguments

Expensive function calls

Time and memory are the two most important resources in computer applications. As a result, an expensive function call is one that consumes large amounts of these two resources due to extensive calculation during execution.

@coder_aishya

Cache

A cache is just a temporary data store(JS object) that stores data in order to serve future requests for that data more quickly.
Note: Browser cache need not be applied in this scenario.

How does it work?

- If the input has been used before, locate it in cache and immediately return the value stored without executing any further computations.
- Use the input to execute any necessary computations, store the results in cache, return the result

@coder_aishya

Steps:

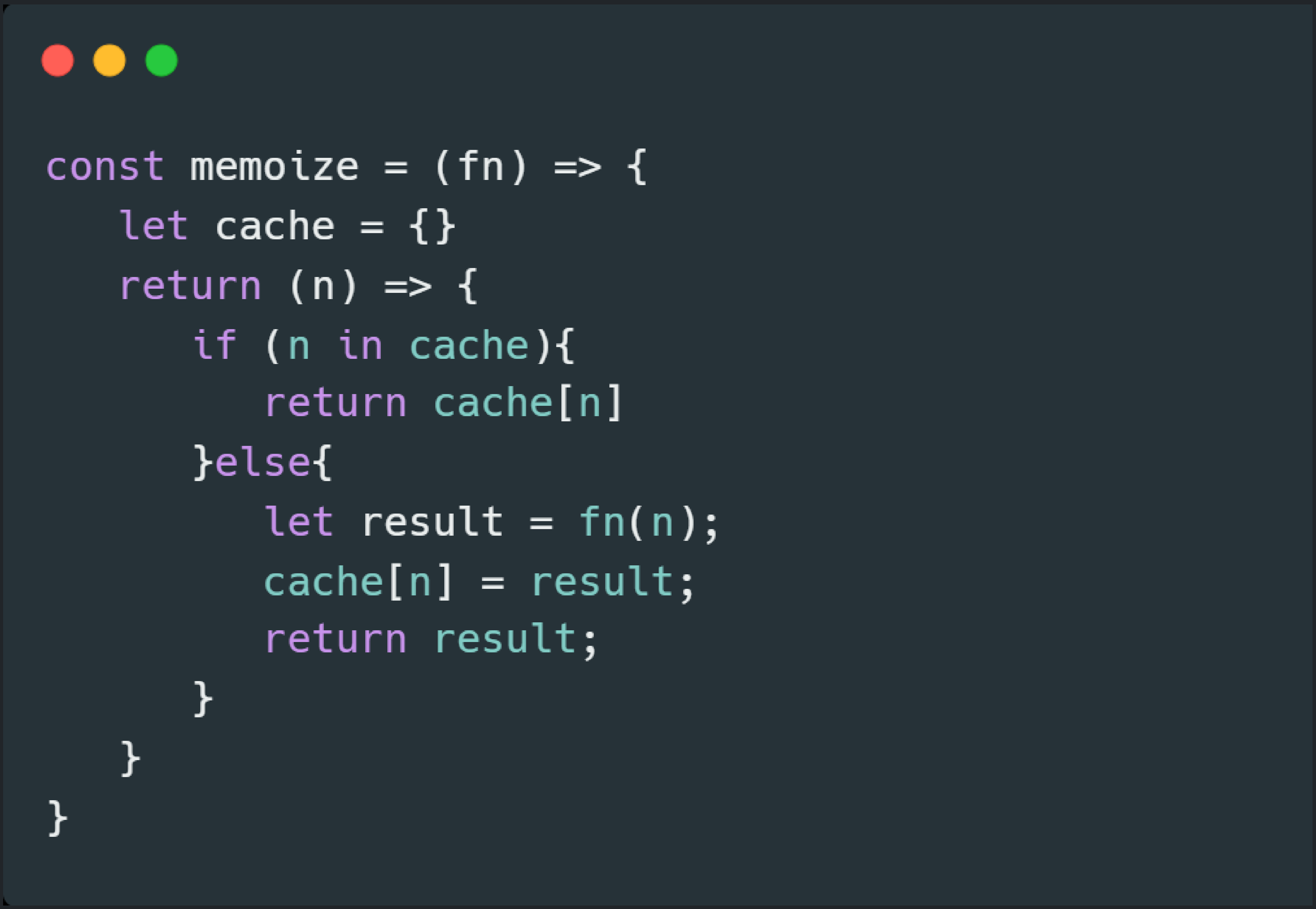
define a function using ES6

```
const memoize = (fn) => {}
```

declare an initial cache object

```
const memoize = (fn) => {
    let cache = {}
}
```

Add logic : If the input is present in cache then return it from cache without computing the same request again else do the computation



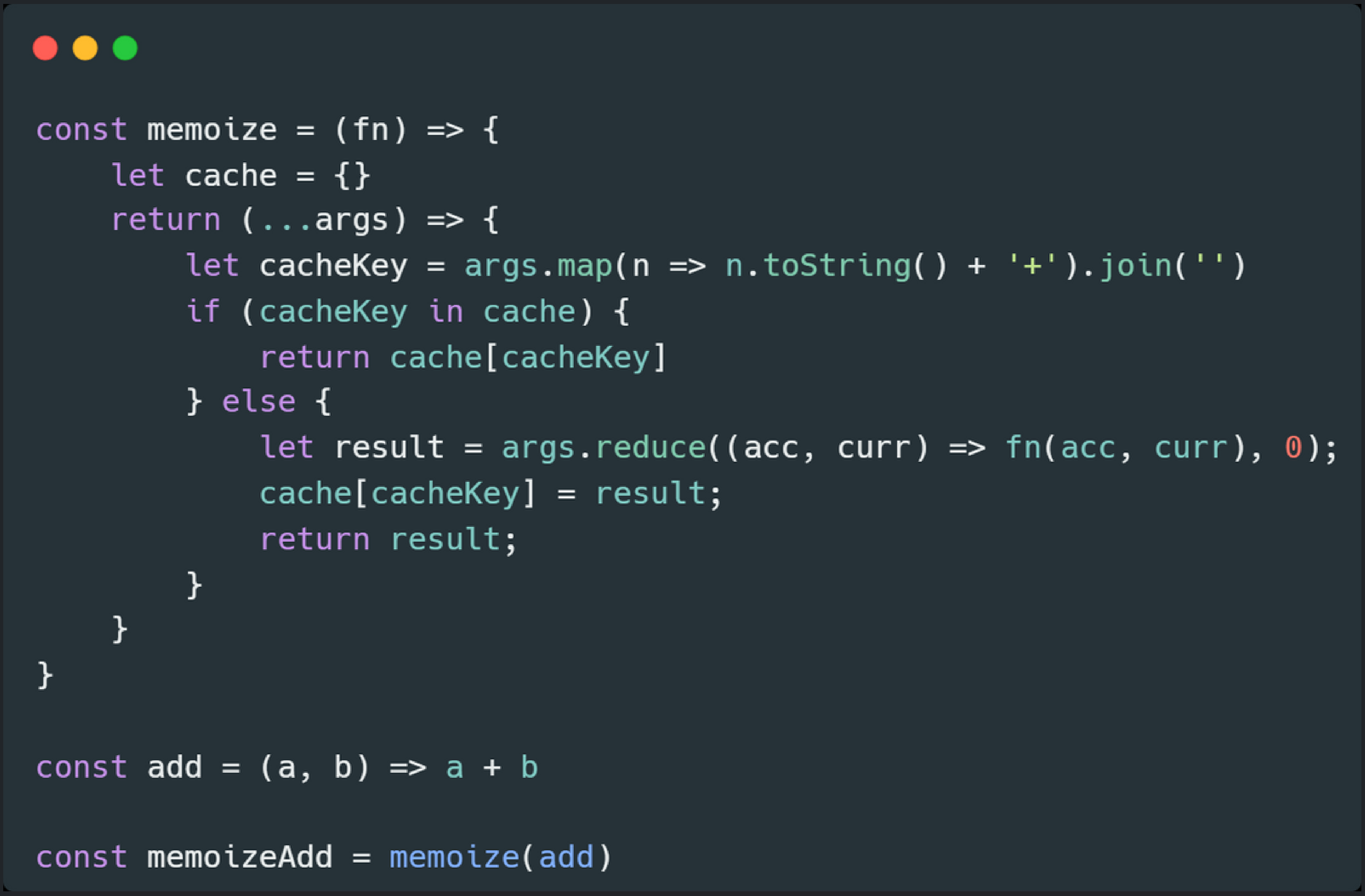
```
const memoize = (fn) => {
    let cache = {}
    return (n) => {
        if (n in cache){
            return cache[n]
        }else{
            let result = fn(n);
            cache[n] = result;
            return result;
        }
    }
}
```

@coder_aishya

Example

Create a memoize function that remembers previous inputs and stores them in cache so that it won’t have to compute the same inputs more than once. The function will take an unspecified number of integer inputs and a reducer method.

Memonizer function



```
const memoize = (fn) => {
    let cache = {}
    return (...args) => {
        let cacheKey = args.map(n => n.toString() + '+').join('')
        if (cacheKey in cache) {
            return cache[cacheKey]
        } else {
            let result = args.reduce((acc, curr) => fn(acc, curr), 0);
            cache[cacheKey] = result;
            return result;
        }
    }
}

const add = (a, b) => a + b

const memoizeAdd = memoize(add)
```

@coder_aishya

Executing the function

```
memoizeAdd(10, 20, 30, 40); //returns 100
memoizeAdd(1, 2, 3, 4); //returns 10
memoizeAdd(5, 10) //returns 15
```

The below statement returns output 10 from cache since its called for the second time

```
memoizeAdd(1, 2, 3, 4); //returns 10
```

Follow me for more

aishwarya-dhuri

coder_aishya