```
const httpOptions = { timeout: 2000, isCache: true };
// later
const { timeout, isCache } = httpOptions;
// you now have a variable named 'timeout'
// and one named 'isCache' with correct values
```

The cool thing is that it also works with nested objects:

```
const httpOptions = { timeout: 2000, cache: { age: 2 } };
// later
const { cache: { age } } = httpOptions;
// you now have a variable named 'age' with value 2
```

And the same is possible with arrays:

```
const timeouts = [1000, 2000, 3000];
// later
const [shortTimeout, mediumTimeout] = timeouts;
// you now have a variable named 'shortTimeout' with value 1000
// and a variable named 'mediumTimeout' with value 2000
```

Of course it also works for arrays in arrays, or arrays in objects, etc.

One interesting use of this can be for multiple return values. Imagine a function `randomPonyInRace` that returns a pony and its position in a race.

```
function randomPonyInRace() {
  const pony = { name: 'Rainbow Dash' };
  const position = 2;
  // ...
  return { pony, position };
}

const { position, pony } = randomPonyInRace();
```

The new destructuring feature is assigning the position returned by the method to the position variable, and the pony to the pony variable! And if you don't care about the position, you can write: