Destructuring assignment

Destructuring assignment

A special syntax to let us unpack either array or object into different variables

Array destructuring

```
let arr = ["Ricky", "Lu"]


let [firstName, lastName] = arr;


console.log(firstName); // "Ricky"
console.log(lastName); // "Lu"
```

The way the destructuring works is that it will assign the first element to the first variable that you gave, second element to the second variable that you gave, and so on...

You can ignore elements that you don't want by not putting a variable for that particular element. For example, if you don't want to assign the second element and the fourth element of the array:

```
let arr = ["Ricky", "Lu", "Xin", "Wang"];


let [first, , third] = arr;


console.log(first); // Ricky
console.log(third); // Xin
```

> This kind of assignment works with any iterable, you can use it on `Set`, and `Map` since destructuring assignment is actually a syntax sugar for calling `for ... of` loops.

You can actually use any assignable on the left side, even object properties!

```
let user = {};
[user.name, user.surname] = ["Ricky", "Lu];
```

Swapping variable

You can use destructuring assignment to swap two variable's value without using an intermediate variable

```
let a = 5;
let b = 100;


[b, a] = [a, b]; // Swaps a's value with b's value
```

The rest '...'

Normally, if you are destructuring say only one value out of the entire array, the rest of the values are just ignored. You can actually gather the rest of the values into a variable as well using the `...` syntax:

```
let [name1, name2, ...rest] = ["Ricky", "Xin", "Rek'sai", "Kai'sa"];


name1 // "Ricky"
name2 // "Xin"
rest // ["Rek'sai", "Kai'sa"]
```

It will fill in the named variable first, then anything that's left over will be stored into the rest variable as an array. Even if there are no items left, the rest variable will remain an empty array.

The rest assignment must also be the last assignment in a destructuring statement!

Default values

When doing destructuring assignment if you use more variable names than the number of elements in the array, the variables that aren't able to receive a value will become `undefined`.

However, you can assign a default value for them if they didn't receive a value from the array.

```
let [firstname="Anonymous", lastname="Anonymous"] = ["Julius"];


firstname // "Julius" because firstname was able to receive a value
lastname // "Anonymous" because there is no more value left for lastname
```

> You can also use function calls for default values. They will only be called if the variable didn't receive a value.

Object destructuring

Just like you can do array destructuring you can also destruct object, the syntax is a little bit different.

```
let {var1, var2} = {var1: ~, var2: ~};
```

If the variable that you are assigning the property of the object into have the same name as the property name then you can just write the property name on the left.

However, if you decide to assign the property to a different variable name say `name` property into just `nm`, then you would have to do a little bit more:

```
let {name: nm, height} = {name: "Ricky", height: "5.9"};
```

You put the variable name that you are actually assigning to on the right side of the `:`, left side is the property name.

You can also use default values as well!

```
let {name: nm, height=500} = {name: "Ricky", age: 50};


// name -> nm
// height -> 500
```

> The order that you put the property assigning does not matter

Rest pattern with object destructuring

Again you can also use the rest variable to assign the rest of the property that you are not assigning directly to a variable into the `rest` variable. The `rest` variable becomes another object with those left over properties that you didn't assign directly.

```
let options = {
title: "Menu",
    height: 200,
    width: 100
};

let {title, ...rest} = options;


// titles -> "Menu"
// rest -> {height: 200, width: 100}
```

Gotcha if there's no `let`

With array destructuring assignment you can declare the variable that you are using for the assigning before you actually do the assigning:

```
let item1, item2;
[item1, item2] = [1];
console.log(item1, item2); // item1 -> 1, item2 -> undefined
```

However, with object destructuring you have to do a little bit more, using the same syntax it would not work!

```
let title, width, height;

// error in this line
{title, width, height} = {title: "Menu", width: 200, height: 100};
```

This is because JavaScript treats `{...}` as a code block, thus it will try to execute it. To fix this you have to wrap the expression in parentheses:

```
let title, width, height;

// error in this line
({title, width, height} = {title: "Menu", width: 200, height: 100});
```

Nested destructuring

If the object that you are destructuring have nested object, then you can extract the nested object's property out as well. You just need to do another layer of destructuring with the same syntax:

```
let nested = {
    another: {
        name: "Inner me",
        age: 30,
        address: "6601231"
    },
    name: "Outer me",
    age: 10
}

let {another: {name: innerName, ...innerRest}, name: outerName, age: outerAge} = nested;
```

```
// innerName -> Inner me
// innerRest -> {age: 30, address: "6601231"}
// outerName -> Outer me
// outerAge -> 10
```

Smart function parameters

If you are going to write a function with many optional parameters, it might looks something like this:

```
function showMenu(title = "Untitled", width = 200, height = 100, items = []) {
  // ...
}
```

Which doesn't look very nice, and when you are calling the function you will have to remember what is what.

To help resolve this we can use destructuring assignment in the function parameter

```
function showMenu({title="Untitled", width=200, height=100, items=[]}) {
// Body of the function
    // title, width, height, items all have default vlaues if you didn't specify it
}

// Calling showMenu
showMenu({}); // With no parameter
showMenu({width: 6969}); // With one optional parameter
```

Now when you are going to call the function, you provide in an object of parameters. If you don't want to provide any then you can just pass in an empty object.

You can make empty object call even better by assigning a default value to the object destructuring assignment.

```
function showMenu({title="Untitled", width=200, height=100, items=[]} = {}) {
// Body of the function
    // title, width, height, items all have default vlaues if you didn't specify it
}

// Calling showMenu
```

```
showMenu(); // With no parameter
showMenu({width: 6969}); // With one optional parameter
```

> Keep in mind that you are still allowed to have other parameter besides the object destructuring assignment

---

Revision #1
Created 2022-12-23 19:08:09 UTC by Tamarine
Updated 2022-12-23 20:56:04 UTC by Tamarine