Chapter 60: Destructuring assignment

Destructuring is a **pattern matching** technique that is added to JavaScript recently in ECMAScript 6.

It allows you to bind a group of variables to a corresponding set of values when their pattern matches to the right hand-side and the left hand-side of the expression.

Section 60.1: Destructuring Objects

Destructuring is a convenient way to extract properties from objects into variables.

Basic syntax:

```
let person = {
  name: 'Bob',
  age: 25
};

let { name, age } = person;

// Is equivalent to
let name = person.name; // 'Bob'
let age = person.age;   // 25
```

Destructuring and renaming:

```
let person = {
  name: 'Bob',
  age: 25
};

let { name: firstName } = person;

// Is equivalent to
let firstName = person.name; // 'Bob'
```

Destructuring with default values:

```
let person = {
  name: 'Bob',
  age: 25
};

let { phone = '123-456-789' } = person;

// Is equivalent to
let phone = person.hasOwnProperty('phone') ? person.phone : '123-456-789'; // '123-456-789'
```

Destructuring and renaming with default values

```
let person = {
  name: 'Bob',
  age: 25
};

let { phone: p = '123-456-789' } = person;
```

```
// Is equivalent to
let p = person.hasOwnProperty('phone') ? person.phone : '123-456-789'; // '123-456-789'
```

Section 60.2: Destructuring function arguments

Pull properties from an object passed into a function. This pattern simulates named parameters instead of relying on argument position.

```
let user = {
    name: 'Jill',
    age: 33,
    profession: 'Pilot'
}

function greeting ({name, profession}) {
    console.log(`Hello, ${name} the ${profession}`)
}

greeting(user)
```

This also works for arrays:

```
let parts = ["Hello", "World!"];

function greeting([first, second]) {
    console.log(`${first} ${second}`);
}
```

Section 60.3: Nested Destructuring

We are not limited to destructuring an object/array, we can destructure a nested object/array.

Nested Object Destructuring

```
var obj = {
  a: {
    c: 1,
    d: 3
  },
  b: 2
};

var {
  a: {
    c: x,
    d: y
  },
  b: z
} = obj;

console.log(x, y, z);     // 1,3,2
```

Nested Array Destructuring

```
var arr = [1, 2, [3, 4], 5];

var [a, , [b, c], d] = arr;
```

```
console.log(a, b, c, d);      // 1 3 4 5
```

Destructuring is not just limited to a single pattern, we can have arrays in it, with n-levels of nesting. Similarly we can destructure arrays with objects and vice-versa.

Arrays Within Object

```
var obj = {
  a: 1,
  b: [2, 3]
};

var {
  a: x1,
  b: [x2, x3]
} = obj;

console.log(x1, x2, x3);    // 1 2 3
```

Objects Within Arrays

```
var arr = [1, 2 , {a : 3}, 4];

var [x1, x2 , {a : x3}, x4] = arr;

console.log(x1, x2, x3, x4);
```

Section 60.4: Destructuring Arrays

```
const myArr = ['one', 'two', 'three']
const [ a, b, c ] = myArr

// a = 'one', b = 'two, c = 'three'
```

We can set default value in destructuring array, see the example of Default Value While Destructuring.

With destructuring array, we can swap the values of 2 variables easily:

```
var a = 1;
var b = 3;

[a, b] = [b, a];
// a = 3, b = 1
```

We can specify empty slots to skip unneeded values:

```
[a, , b] = [1, 2, 3] // a = 1, b = 3
```

Section 60.5: Destructuring inside variables

Aside from destructuring objects into function arguments, you can use them inside variable declarations as follows:

```
const person = {
  name: 'John Doe',
  age: 45,
  location: 'Paris, France',
};
```

```
let { name, age, location } = person;

console.log('I am ' + name + ', aged ' + age + ' and living in ' + location + '.');
// -> "I am John Doe aged 45 and living in Paris, France."
```

As you can see, three new variables were created: `name`, `age` and `location` and their values were grabbed from the object `person` if they matched key names.

Section 60.6: Default Value While Destructuring

We often encounter a situation where a property we're trying to extract doesn't exist in the object/array, resulting in a `TypeError` (while destructuring nested objects) or being set to **`undefined`**. While destructuring we can set a default value, which it will fallback to, in case of it not being found in the object.

```
var obj = {a : 1};
var {a : x , b : x1 = 10} = obj;
console.log(x, x1); // 1, 10

var arr = [];
var [a = 5, b = 10, c] = arr;
console.log(a, b, c); // 5, 10, undefined
```

Section 60.7: Renaming Variables While Destructuring

Destructuring allows us to refer to one key in an object, but declare it as a variable with a different name. The syntax looks like the key-value syntax for a normal JavaScript object.

```
let user = {
  name: 'John Smith',
  id: 10,
  email: 'johns@workcorp.com',
};

let {user: userName, id: userId} = user;

console.log(userName) // John Smith
console.log(userId) // 10
```