Spread operator (…)

What does it look like? Three dots: **...**
What does it do? The spread operator allows an expression to be expanded in places where multiple elements/variables/arguments are expected.

Let's have a closer look into this, and see where it can be useful:

- Replace apply/function calls
- Useful when manipulating Arrays
- Spread in Object literals

1. Replace "apply" function/used in function calls

So, if we have a function that takes three arguments, like this:

```
const fullName = (firstName, middleName, lastName) => firstName + middleName + lastName;
```

And we have a list of names stored in an Array structure:

```
const names = ["Marry", "Anne", "Andersen"];
```

Now if we want to invoke fullName function we should do something like this:

```
fullName(names[0], names[1], names[2]);
```

...or using apply function, like this:

```
fullName.apply(this, names);
```

Here is where Spread operator comes to the rescue:

```
fullName(...names)
```

So what happens here is that names array is **spread** to multiple arguments.

2. Useful when manipulating Arrays

- It's easier to create a new Array that contains the elements from another array

```
var parts = ['shoulders', 'knees'];
var lyrics = ['head', ...parts, 'and', 'toes'];
```

- It's easier to concatenate Arrays

The old fashioned way:

```
var arr1 = [0, 1, 2];
var arr2 = [3, 4, 5];
// Append all items from arr2 onto arr1
arr1 = arr1.concat(arr2);
```

...nowdays

```
var arr1 = [0, 1, 2];
var arr2 = [3, 4, 5];
arr1 = [...arr1, ...arr2];
```

- Copying another array's elements has never been easier

```
var arr = [1, 2, 3];
var arr2 = [...arr];
arr2.push(4);

// arr2 becomes [1, 2, 3, 4]
// arr remains unaffected
```

3. Spread in Object literals(JS objects {}) - Draft

```
var obj1 = { foo: 'bar', x: 42 };
var obj2 = { foo: 'baz', y: 13 };

var clonedObj = { ...obj1 };
// Object { foo: "bar", x: 42 }

var mergedObj = { ...obj1, ...obj2 };
// Object { foo: "baz", x: 42, y: 13 }
```

[Read more]
http://jpsierens.com/three-dots-javascript-spread-operator/
https://davidwalsh.name/spread-operator

Rest parameters

The rest parameter syntax allows us to represent an indefinite number of arguments as an array.

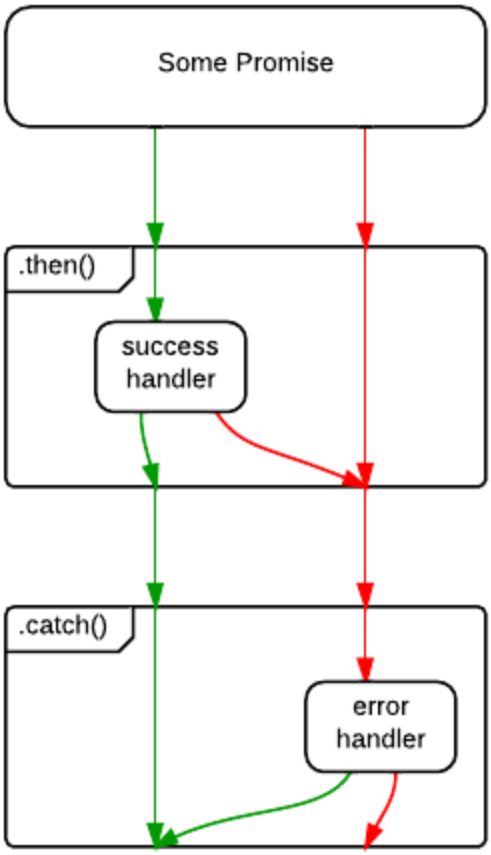
```
function sum(…numbers) {
  var result = 0;
  numbers.forEach(function (number) {
    result += number;
  });
  return result;
}
console.log(sum(1)); // 1
console.log(sum(1, 2, 3, 4, 5)); // 15
```

Promises, async and await

We use promises whenever there is an async operation to be done, and we want to process the data only when we are sure the operation is done with success, or do some error handling if the operation fails.

A promise may be in one of 3 possible states: fulfilled, rejected, or pending. Promise users can attach callbacks to handle the fulfilled value or the reason for rejection.

E.g: we use Promises to make server requests.



Async and await

Any promise we have, using ES2016, we can await. That's literally all await means: it functions in exactly the same way as calling `.then()` on a promise (but without requiring any callback function)

Using promises:

```
function getFirstUser() {
    return getUsers().then(function(users) {
        return users[0].name;
    }).catch(function(err) {
        return {
          name: 'default user'
        };
    });
}
```

Using async/await:

```
async function getFirstUser() {
    try {
        let users = await getUsers();
        return users[0].name;
    } catch (err) {
        return {
            name: 'default user'
        };
    }
}
```

[Read more]
https://medium.com/javascript-scene/master-the-javascript-interview-what-is-a-promise-27fc71e77261
https://medium.com/@bluepnume/learn-about-promises-before-you-start-using-async-await-eb148164a9c8