JavaScript: a smarter approach to error handling in functions with the Error object

In JavaScript we can implement alternative error handling in functions using their return values.

We usually use a default value or raise an exception when an error occurs.

```
'use strict';

const myFunctionWithThrow = dataObject => {
    if(typeof dataObject !== 'object') {
        throw new Error('Invalid argument.');
    }
};

const myFunctionWithValue = dataObject => {
    if(typeof dataObject !== 'object') {
        return null;
    }
};
```

In the first case, the code execution is interrupted, and in the second case we have a `null` value which in fact cannot be reused apart from the verification that we carry out when we execute the function and which above all does not contain any information.

We can instead use the `Error` object differently: instead of throwing an exception and blocking all the following code, we can use it as the return value of the function.

```
'use strict';

const myFunction = dataObject => {
    if(typeof dataObject !== 'object') {
        return new Error('Invalid argument.');
    }
};
```

So we can perform the following verification:

```
'use strict';

const result = myFunction();

if(result instanceof Error) {
    // Error handling
}
```

In this way we do not block the flow of code and we have a semantic value that we can reuse, for example in our logging system.