Object Destructuring

Doesn't matter if we create object directly or we use classes, eventually the object created is a combination of key value pairs.

Object destructuring is a mechanism using which we try to unpack the object into multiple variables.

In an object, combination of key value pairs are associated to one entity, but using de-structuring we can associate them to individual separate variables.
\

```
const product = {name: "Iphone 14 prp", price: 125000, category: "Mobiles"};

const {name, price, category} = product;
```

Here, we are wrapping variable names inside a pair of curly braces and equating it to the object, this unpacks the object and put individual key value pair values in these variables. Here we have to give exact name of the keys as the variable names, like we cannot directly say productName instead of name.

```
const {productName} = product; // this will put undefined inside productName
```

Also, it is not mandatory to unpack all the key value pairs, if we just want a few of them, this syntax still works.

To give original keys some other names in the destructured variables, we can use alias.
To put an alias, we just say { originalName: alias} while destructuring objects.

```
const {name: productName, price: productPrice, category } = product;
```

Here value of the name key is actually put inside a variable with name `productName`, for price key it is put inside `productPrice` and for category we don't have an alias hence variable name is category. So technically productName and productPrice are alias for name and price.

Destructuring an object inside another object

Sometimes, we want to create new object using key value pairs of another object, in that case we can use spread operator: `...` that can help us to unpack one object's all key value pairs inside another one.

```
const product = {name: "Iphone 14 prp", price: 125000, category: "Mobiles"};

const purchasedProduct = {orderId: "xsya122", orderDate: "11/09/2024",
 ...product }
```

Here in the `purchasedProduct` object, it has a few of it's own key value pairs and then all the key value pairs from the product object is unpacked inside it.

So purchasedProduct looks like this:

```
{orderId: 'xsya122', orderDate: '11/09/2024', name: 'Iphone 14 prp', price:
125000, category: 'Mobiles'}
```

What to do for nested object ?

Say we have a nested object like this:

```
const product = {name: "Iphone 14 prp", price: 125000, category: {name:
"Mobiles", categoryId: 12} };
```

We want to unpack the categoryId key which is present in a nested object category. One way will be to first unpack. category in a variable and then unpack categoryId.

```
const { category } = product;
const {categoryId} = category;
```

But we have a one liner way to do the same thing

```
const { category: { categoryId } } = product
```

This sntax will only fetch categoryId for you and nothing else.

Tips for object destructuring:

- If we need to combine multiple objects into a new one we can use spread operator:

```
const result = {...o1, ...o2, ...o3}; // this will combine o1, o2, o3
```

- If we need to clone an object, we can again use spread operator

```
const clone = {...o1}; // created clone of o1 as clone variable
```

- If we want to update value of an object and create a new one also at the same time we can again use spread operator

```
const clone = {..o1, keyToBeUpdate: value};
```

- We can give default values also while destructuring objects.

```
const {name, discount = 10} = product
```

- In the above code, if product does't contain a discount key, then discount variable will get a value 10, else whatever is the value of discount key in the product will be given to the discount variable.

Array destructuring

- All the destructuring logic works well for arrays also, it's just that instead of pair of curly braces, to handle arrays we use square brackets.
- For arrays, names, don't matter, only sequencing does. So while destructuring we can give any names to the variables and sequentially they will be allocated.

```
const [english, hindi, maths, science, sst] = [100, 100, 98, 99, 100];
```

- Here we get 5 variables, where english, hindi and sst has values 100, science is 99 and maths is 98.
- Just like objects, if we don't wahnt to allocate all the values in separate variables then also it can be done.

```
const [english, hindi, maths] = [100, 100, 98, 99, 100];
```

- Here only first three values are allocated to english hindi and maths, rest are kept as it is.

- Just like using spread operator we can unpack an object inside another object, we can do same for arrays.

```
const x = [1,2,3];
const y = [4,5,6, ...x];// [4,5,6,1,2,3]
```

Rest parameter

The three dot symbol `...` can be used as spread operator or rest parameter. But what is rest parameter ?
Rest parameter is opposite of spread operator. Spread operator unpacks key value pairs, where as rest parameter pack them up in one unit.

```
const {discount, ...productWithoutDiscount} = product;
```

Here product object's discount key will not be part of `productWithoutDiscount` variable. In the `productWithoutDiscount` variable everything else coming from an object is combined into one and stored.

The rest parameter syntax also allows a function to accept an indefinite number of arguments as an array, providing a way to represent variadic functions in JavaScript.

```
function sum(...theArgs) {
  let total = 0;
  for (const arg of theArgs) {
    total += arg;
  }
  return total;
}

console.log(sum(1, 2, 3));
// Expected output: 6

console.log(sum(1, 2, 3, 4));
// Expected output: 10
```

Here the theArgs variable is technically an array which is combining multiple arguments into one.