↱) ~~[illegible]~~ Currying in JS

Functional Programming technique → where → Multi argument function → transformed → into → Series → of → Single argument functions

facilitating → Partial Applications
→ Composability

↱) Key Points

→ Enables → Partial applications $\xrightarrow{\text{of}}$ arguments.
→ Enhances → composability → through → functional chaining
→ Utilizes → Function closures, ↳ Higher order Function } → for → implementation.

↱) Original Function with multiple arguments

```
<script>
    function filterByProperty(array , property, value){
        return array.filter(item => item[property] === value);
    }
</script>
```

↱) Curried Version

```
<script>
    function filterByProperty (property){
        return function(value){
            return function(array){
                array.filter(item => item[property] === value))
            };
        };
    }
<script>
```

Case 1〉

```
Const filterByGender = curiedfilterByProperty('gender');
const filterByNationality = curriedfilterByProperty('nation');
```

Case1

```
const UKusers = filterByNationality ('UK')(users);
for (let val in UKusers){
    col1.innerHTML += `UK users: ${UKusers[val].name}`+
}
```

O/p :

~~UKuser~~

Curred version with case 1

```
UKusers : Charlotte        UK users: Manjula
UKusers; Mandy
~~UK user: [illegible]~~
```

Working

f{ } —UK→ {f{ }} —users→ {f{ }}→

Case 2

```
const femaleusers = filterByGender('F')(users);
for(let val in UKusers){
    co2.innerHTML += `Female user: ${femaleusers[val].name}`+"<br/>";
}
```

Curried Version with case 2

```
Female user: Charlotte
Female user: Mandy
Female user: Manjola
```