"Data Structure and modern operators"

* Destructuring arrays: →

```
const arr = [2, 3, 4];
const a = arr[0];
const b = arr[1];
const c = arr[2];

const [x, y, z] = arr;
console.log(x, y, z);
console.log(arr);
```

* Destructuring the array like this doesn't affect the original array, it simply assign the numbers of arr to the variables x, y, z.

```
// Switch Through Destructuring
let [main, secondary] = restaurant.starterMenu;
[secondary, main] = [main, secondary];
console.log(main, secondary);
```

* Switch two elements throught Destructuring

* Also you can use this to receive two elements from a function at once.

```
// Nested destructuring
const nested = [2, 4, [5, 6]];
// const [i, , j] = nested;
const [i, , [j, k]] = nested;
console.log(i, j, k);
```

This is called a destructure of a destructure (nested).

* Destructuring objects:→

* in it you have to specify the same name of the properties in the original object, as the order of properties in object doesnt matter.

```
const { name, openingHours, categories } = restaurant;
console.log(name, openingHours, categories);
```

* if I want to change this original name How it will be done?

* note that:- order of properties doesnt matter but its name should be identical.

```
const {name: nameNew, openingHours: openingHoursNew, categories:categoriesNew} = restaurant;
console.log(nameNew, openingHoursNew, categoriesNew);
```

↳ This is exactly how I change names in case of objects.

```
// Default values
const { menu = [], starterMenu: starters = [] } =
restaurant;
console.log(menu, starters);
```

assign default values in case it doesnt exist

* variable mutating in objects (switching):→

```
let a = 20;
let b = 30;
const obj = { a: 2, b: 3, c: 4 };
({ a, b } = obj);
console.log(a, b);
```

```
// Nested objects
const {
  fri: { open: o, close: c },
} = openingHours;
console.log(o, c);
```

if I want to get the values of open and close.

```
restaurant {
        openingHours {
            Fri { open: '10', close: '22'
                }
        }
    };
```

* The spread operator :→

```
const arr = [7, 8, 9];
const badNewArr = [1, 2, arr[0], arr[1], arr[2]];
console.log(badNewArr);

const newArr = [1, 2, ...arr];
console.log(newArr);
```

* It is used to extract the elements of arr to the newArr array individual

* In anyplace when you want to get element separated by , the spread operator will be your best.

* iterables :- arrays, string, maps, sets, but not object
spread operator can be used there.

```
// Iterables: arrays, strings, maps, sets. NOT objects
const str = 'Jonas';
const letters = [...str, ' ', 'S.'];
console.log(letters);
```

* note that :→ Remember when we make a copy of an object using this way const newobj = obj;
if I change any thing in the new obj this will be reflected to the original one.

* But in the case of spread operator it makes new object with new address and assign the old one to it.

```
const restaurantCopy = { ...restaurant };
restaurantCopy.name = 'Ristorante Roma';
console.log(restaurantCopy.name);
console.log(restaurant.name);
```

So if I change any of the copy that does not get reflected to the original one.

* Rest Pattern and Parameters :→

Rest is used to pack individual elements to an array unlike spread operators which used to unpack.

note that :→ spread operator lands on the Right side of equal and used to unpack elements into individuals.

```
// SPREAD, because on RIGHT side of =
const arr = [1, 2, ...[3, 4]];

// REST, because on LEFT side of =
const [a, b, ...others] = [1, 2, 3, 4, 5];
console.log(a, b, others);
```

↳ will print 3,4,5

note also that :→ Rest operator lands on the left side of the assignment operator and means get the Rest of elements after the last selected.

```
// 2) Functions
const add = function (...numbers) {
  console.log(numbers);
};

add(2, 3);
add(5, 3, 7, 2);
add(8, 2, 5, 3, 2, 1, 4);
```

we pass individual values to the Function and it tries to pack them into an array.

* Short Circuting: →

11) oR operator simply accepts any dataTypes and can return any dataType.

```
console.log(3 || 'Jonas');
console.log('' || 'Jonas');
console.log(true || 0);
console.log(undefined || null);
```

→ 3
→ Jonas
→ True
→ null

* The mean of short circuting is that if it faces the first Truthy value then, there is no need to proceed and simply returns the first truthy value.

88)

```
console.log('---- AND ----');
console.log(0 && 'Jonas');
console.log(7 && 'Jonas');

console.log('Hello' && 23 && null && 'jonas');
```

→ 0
→ Jonas
→ null

* and operator short circuting at the first falsy value and returns it.

* unlike the oR operator short circuting at the first truthy value and simply returns it.

* Nullish Coalising operator :- ??

```
// Nullish: null and undefined (NOT 0 or '')
const guestCorrect = restaurant.numGuests ?? 10;
console.log(guestCorrect);
```

* it simply means that the process will continue execut if the first part is null or undefined.

* if numGuests doest exist as a property in Restaurant object, the guss correct will be 10.

* if it exists then nothing will change.

* The For of loop :-

```
for (const item of menu) console.log(item);
```

each item represent elemt

```
for (const [i, el] of menu.entries()) {
  console.log(`${i + 1}: ${el}`);
}
```

If I wanna get the index of the curr elm, simply entities give us [itr, elm] which we destructure it.

* Enhanced object literal :→

1) There is no need to make property in object and assign function expression to it like

```
order: function (starterIndex, mainIndex) {
  return [this.starterMenu[starterIndex], this.
  mainMenu[mainIndex]];
},
```

instead →

```
order(starterIndex, mainIndex) {
  return [this.starterMenu[starterIndex], this.
  mainMenu[mainIndex]];
},
```

and the order will represent the property.

2) The property can be assigned to an expression like this :→

```
const weekdays = ['mon', 'tue', 'wed', 'thu', 'fri',
'sat', 'sun'];
const openingHours = {
  [weekdays[3]]: {
    open: 12,
    close: 22,
  },
  [weekdays[4]]: {
    open: 11,
    close: 23,
  },
  [`day-${2 + 4}`]: {
    open: 0, // Open 24 hours
    close: 24,
  },
};
```

↳ dont forget the square brackets at the property name.

* optional chaining :→

```
if (restaurant.openingHours && restaurant.
openingHours.mon)
  console.log(restaurant.openingHours.mon.open);

// WITH optional chaining
console.log(restaurant.openingHours.mon?.open);
```

it means that if one of the properties doesnt exist it simply stop the chaining and returns undefined so that, we don't get into the case where

undefined.property name

which will lead to an error.

? ↳ it simply checks if the part on the left exist or not, if is exist it proceed executing otherwise it returns undefined.

```
const properties = Object.keys(openingHours);
console.log(properties);

let openStr = `We are open on ${properties.length}
days: `;
for (const day of properties) {
  openStr += `${day}, `;
}
console.log(openStr);

// Property VALUES
const values = Object.values(openingHours);
console.log(values);

// Entire object
const entries = Object.entries(openingHours);
console.log(entries);
```

* keys gives us array of keys only.

entries gives us a complete object devided.

* Such as this and you can continue destructuring.

```
[
  [ 'thu', { open: 12, close: 22 } ],
  [ 'fri', { open: 11, close: 23 } ],
  [ 'sat', { open: 0, close: 24 } ]
]
```

* Sets :→

Sets is like an array but there is two main difference
1- order of elemets doesnt matter like array.
2- There is no duplicates.

Create Sets :→

```
const ordersSet = new Set([
  'Pasta',
  'Pizza',
  'Pizza',
  'Risotto',
  'Pasta',
  'Pizza',
]);
```

more Methods :→

```
console.log(ordersSet.size);
console.log(ordersSet.has('Pizza'));
console.log(ordersSet.has('Bread'));
ordersSet.add('Garlic Bread');
ordersSet.add('Garlic Bread');
ordersSet.delete('Risotto');
```

* maps :- maps is used to pair specific key with value

How to create a map!

```
const rest = new Map();
rest.set('name', 'Classico Italiano');
rest.set(1, 'Firenze, Italy');
console.log(rest.set(2, 'Lisbon, Portugal'));
```

Note that :- When we set in map it Basically add a new key paired with value and returns the updated version of the map.

* maps is simply an Array of Array like this :→

```
const question = new Map([
  ['question', 'What is the best programming language
  in the world?'],
  [1, 'C'],
  [2, 'Java'],
  [3, 'JavaScript'],
  ['correct', 3],
  [true, 'Correct 🎉'],
  [false, 'Try again!'],
]);
console.log(question);
```

and object.entries() → returns an array of Array hence we can convert the Res of object.entries into a map

Const newobj = new Map(object.entries());

↳ will be a map (Array of array).

"Working with Strings"

Reminder:→ you know that all strings are Primitive data Types, This mean it doesn't have methode like objects, True?

```
const checkMiddleSeat = function (seat) {
  // B and E are the middle Seats
  const lastChar = seat.slice(-1);
  if (lastChar === "B" || lastChar === "E") return "You Got The middle Seat";
  else "You Are lucky not have middle seat";
};

console.log(checkMiddleSeat("11B"));
checkMiddleSeat("11C");
```

So How in this slice method Being called on a Primitive data Type?

The answer is that:→

① JS Converts this String to a new object like that

Const newString = new String(old String);

② Then we can apply these Methods upon an object.

methods:→

① Trim:— Used to remove white spaces from string it has also Trimstart and Trim end which removes white space in the begin or end.

② Replace:—

```
// replacing
const priceGB = '288,97£';
const priceUS = priceGB.replace('£', '$')
console.log(priceUS);
```

③ Split:→

```
console.log('a+very+nice+string'.split('+'));
console.log('Jonas Schmedtmann'.split(' '));

const [firstName, lastName] = 'Jonas Schmedtmann'.
split(' ');
```

Split the string Based on arguments and returns the new in array.

④ Join:→

The Reverse of split method is Join which used to Concatenate array elements Based upon certain mark.

```
const fullName = ["Engineer.", firstName, lastName.toUpperCase()].join(" ");
```

This means it will be a spaces Between MR, The first and last name

⑤ Padding :→

```
// Padding
const message = 'Go to gate 23!';
console.log(message.padStart(25, '+').padEnd(30, '+'))
;
console.log('Jonas'.padStart(25, '+'));
```

① PadStart(final length after Padding, character you want apply padding with);