ES+

tsevdos.me / @tsevdos

Rules

Feel free to interrupt me for:

- questions
- relevant comments

Agenda

- template literals
- shorthand property names
- arrow functions
- destructuring assignment (objects and arrays)
- function parameters destructuring
- default parameters

Agenda

- spread and rest
- ES modules
- array methods
  - (map, filter, reduce, includes, find, some, every)
- operators:
  - conditional (ternary) operator
  - optional chaining operator
  - logical `AND` operator (`&&`)
  - logical `OR` operator (`||`)
  - nullish coalescing operator

Template literals

```
const greeting = "Hello";
const subject = "World";

console.log(`${greeting} ${subject}!`); // Hello World!

// same as:
console.log(greeting + " " + subject + "!");
```

Shorthand property names

```
const name = "John";
const lastName = "Doe";
const technologies = ["JavaScript", "TypeScript", "React", "Nextjs"];

console.log({ name, lastName, technologies });

// same as:
console.log({ name: name, lastName: lastName, technologies: technologies });
```

Arrow functions

- aka. arrow function expression
- always anonymous
- don’t have bindings to `arguments`, `this` and `super`, and cannot be used as methods
- cannot be used as constructors (calling them with `new` throws a `TypeError`)
- implicit return

Arrow functions

```
const add = (number1, number2) => number1 + number2;
const getFive = () => 5;

console.log(add(2, 3));
console.log(getFive());

// same as:
function add(number1, number2) {
  return number1 + number2;
}

function getFive() {
  return 5;
}
```

Object destructuring

```
const user = {
  name: "John",
  lastName: "Doe",
  technologies: ["JavaScript", "TypeScript", "React", "Nextjs"],
};
const { name, lastName, technologies } = user;

console.log(name);
console.log(lastName);
console.log(technologies);

// same as:
const name = user.name;
const lastName = user.lastName;
const technologies = user.technologies;
```

Object destructuring (nested objects)

```
const user = {
  name: "John",
  lastName: "Doe",
  address: {
    city: "Athens",
    street: "Ermou",
    number: 10,
    country: "Greece",
  },
};

const {
  name,
  address: { city, country },
} = user;

console.log(name);
console.log(city);
console.log(country);

// same as:
const name = user.name;
const city = user.address.city;
const country = user.address.country;
```

Object destructuring (default values)

```
const user = {
  name: "John",
  lastName: "Doe",
};

const { name = "stranger", lastName = null, age = 18 } = user;

console.log(name);
console.log(lastName);
console.log(age);

// same as:
const name = user.name || "stranger";
const lastName = user.lastName || null;
const age = user.age || 18;
```

Object destructuring (rename variables)

```
const user = {
  name: "John",
  lastName: "Doe",
};

const { name: myName, lastName: myLastName } = user;

console.log(myName);
console.log(myLastName);

// same as:
const myName = user.name;
const myLastName = user.lastName;
```

Object destructuring (rename variables and default values)

```
const user = {
  name: "John",
  lastName: "Doe",
};

const { name: myName = "stranger", lastName: myLastName = null, age: myAge = 18 } = user;

console.log(myName);
console.log(myLastName);
console.log(myAge);

// same as:
const myName = user.name || "stranger";
const myLastName = user.lastName || null;
const myAge = user.age || 18;
```

Array destructuring

```
const languages = ["JavaScript", "TypeScript", "Rust"];

const [js, ts, rs] = languages;

console.log(js);
console.log(ts);
console.log(rs);

// same as:
const js = languages[0];
const ts = languages[1];
const rs = languages[2];
```

Array destructuring (ignoring values)

```
const languages = ["JavaScript", "TypeScript", "Rust"];

const [js, , rs] = languages;

console.log(js);
console.log(rs);

// same as:
// const js = languages[0];
// const rs = languages[2];
```

Array destructuring (default values)

```
const languages = ["JavaScript", , "Rust"];

const [js, ts = "TypeScript", rs, go = "Go"] = languages;

console.log(js);
console.log(ts);
console.log(rs);
console.log(go);

// same as:
const js = languages[0];
const ts = languages[1] || "TypeScript";
const rs = languages[2];
const go = languages[3] || "Go";
```

Function parameters destructuring (object)

```
const user = {
  name: "John",
  lastName: "Doe",
};

function fullname({ name, lastName }) {
  console.log(`Hello, ${name} ${lastName}!`);
}

fullname(user);
```

Function parameters destructuring (object - rename variables)

```
const user = {
  name: "John",
  lastName: "Doe",
};

function fullname({ name: userName, lastName: userLastName }) {
  console.log(`Hello, ${userName} ${userLastName}!`);
}

fullname(user);
```

Function parameters destructuring (object - default values)

```
const user = {
  name: "John",
  lastName: "Doe",
};

function fullname({ name = "stranger", lastName = "", age = 18 }) {
  console.log(`${name} ${lastName} is ${age}!`);
}

fullname(user);
```

Function parameters destructuring (array)

```
const languages = ["JavaScript", "TypeScript", "Rust"];

function logMyLanguages([js, , rs]) {
  console.log(`${js} - ${rs}`);
}

logMyLanguages(languages);
```

Function parameters destructuring (array - default values)

```
const languages = ["JavaScript", , "Rust"];

function logMyLanguages([js, ts = "TypeScript", rs, go = "Go"]) {
  console.log(`${js} - ${ts} - ${rs} - ${go}`);
}

logMyLanguages(languages);
```

Default parameters

```
const sayHello = (salut = "Hello", name = "stranger") => {
  return `${salut} ${name}!`;
};

console.log(sayHello()); // "Hello stranger!"
console.log(sayHello("Hi", "John")); // "Hi John!"
console.log(sayHello(undefined, "World")); // "Hello World!"
```

Object spread

```
const user = {
  name: "John",
  lastName: "Doe",
};

const reactDeveloper = {
  skills: ["JavaScript", "React"],
  OOP: true,
};

const userDev = { ...user, ...reactDeveloper };

console.log(userDev);
```

Object rest

```
const user = {
  name: "John",
  lastName: "Doe",
  age: 25,
};

const { age, ...rest } = user;

console.log(age); // 25
console.log(rest); // { name: "John", lastName: "Doe" }
```

Array spread

```
const favoriteLanguages = ["JavaScript", "TypeScript"];
const learningLanguages = ["Rust", "Go"];

const allLanguages = [...favoriteLanguages, ...learningLanguages];

console.log(allLanguages); // ["JavaScript", "TypeScript", "Rust", "Go"]
```

Array rest

```
const languages = ["JavaScript", "TypeScript", "Rust", "Go"];
const [js, ...rest] = languages;

console.log(js); // "JavaScript"
console.log(rest); // ["TypeScript", "Rust", "Go"]
```

JavaScript modules

- AMD (asynchronous module definition)
- CommonJS (node.js)
- ES modules (ES2015 / ES6)

ES modules

- reusability (code is organized into reusable, self-contained modules)
- break code into smaller parts, making it easier to manage, debug, and maintain
- namespace, encapsulation and isolation (each module has its own scope)
- 3rd party packages (npm)
- explicit dependencies (clearly state their dependencies)

ES modules: export and import types

- classes
- functions
- variables
  - object
  - array
  - string
  - number
  - boolean
  - everything that can be stored into a variable

ES modules can export:

- **only one default** export
- **as many as you want named** exports

ES modules: default export

- export

```
export default function add(a, b) {
  return a + b;
}
```

- import

```
import add from "./module.js";
import myAdd from "./module.js";
```

ES modules: named export

- export

```
export const subtract = (a, b) => a - b;
export const user = { name: "John", lastName: "Doe" };
export const languages = ["JavaScript", "TypeScript", "Rust"];
```

- import

```
import * as ModuleName from "./module.js"; // ModuleName.subtract
import { subtract, user, languages } from "./module.js";
import { subtract as muSub, user as john, languages } from "./module.js";
```

ES modules: default and named export

- export

```
export default function add(a, b) {
  return a + b;
}

export const user = { name: "John", lastName: "Doe" };
export const languages = ["JavaScript", "TypeScript", "Rust"];
```

- import

```
import add, { user, languages } from "./module.js";
import myAdd, { user as john, languages } from "./module.js";
```

ES modules: re-exports

- export

```
// index.js
export { default as formData } from "./form/form-data.js";
export { minlength, maxlength } from "./form/validations.js";
```

- import

```
import { formData, minlength, maxlength } from "./index.js";
```

Array methods: map

```
const languages = ["JavaScript", "TypeScript", "Rust", "Go"];

const noOfLetters = languages.map((language, index, arr) => {
  return {
    [index]: language.length,
  };
});

console.log(noOfLetters);
```

Array methods: filter

```
const numbers = [0, 1, 2, 3, 5, 8, 13, 21];
const largerThanFive = numbers.filter((num) => num > 5);

console.log(largerThanFive);
```

Array methods: reduce

```
const numbers = [0, 1, 2, 3, 5, 8, 13, 21];
const sum = numbers.reduce((acc, num) => acc + num, 0);

console.log(sum);
```

Array methods: includes

```
const languages = ["JavaScript", "TypeScript", "Rust", "Go"];
const numbers = [0, 1, 2, 3, 5, 8, 13, 21];

const knowsTypeScript = languages.includes("TypeScript");
const includesNumberTen = numbers.includes(10);

console.log(knowsTypeScript);
console.log(includesNumberTen);
```

Array methods: find

```
const users = [
  { name: "John", age: 25 },
  { name: "Jane", age: 30 },
  { name: "Jack", age: 35 },
];
const john = users.find(({ name }) => name === "John");
const olderUser = users.find(({ age }) => age > 33);

console.log(john);
console.log(olderUser);
```

Array methods: some

```
const users = [
  { name: "John", age: 25 },
  { name: "Jane", age: 30 },
  { name: "Jack", age: 35 },
];
const someAreYoung = users.some(({ age }) => age < 26);
const someAreJane = users.some(({ name }) => name === "Jane");

console.log(someAreYoung);
console.log(someAreJane);
```

Array methods: every

```
const users = [
  { name: "John", age: 25, isDeveloper: true },
  { name: "Jane", age: 30, isDeveloper: true },
  { name: "Jack", age: 35, isDeveloper: true },
];
const allAreYoung = users.every(({ age }) => age < 26);
const allAreDevs = users.every(({ isDeveloper }) => isDeveloper);

console.log(allAreYoung);
console.log(allAreDevs);
```

Conditional (ternary) operator

```
// condition ? val1 : val2;

const ageGroup = age >= 18 ? "adult" : "kid";
console.log(ageGroup);
```

Optional chaining operator

```
const user = {
  name: "John",
  lastName: "Doe",
  address: {
    city: "Athens",
    street: "Ermou",
    country: "Greece",
  },
};
const city = user?.address?.city;

console.log(city);

// same as:
// const city = user && user.address && user.address.city;
```

Logical `AND` operator (`&&`)

```
// expr1 && expr2;

console.log(true && true); // true
console.log(true && false); // false
console.log(true && "My name!"); // "My name!"
console.log(true && fn()); // runs fn!
```

Logical `OR` operator (`||`)

```
// expr1 || expr2;

console.log(true || true); // true
console.log(true || false); // true
console.log(false || false); // false
console.log("My name!" || false); // "My name!"
```

Nullish coalescing operator

- returns its right-hand side operand when its left-hand side operand is `null` or `undefined`, and otherwise returns its left-hand side operand

```
// undefined ?? 0; // 0
// null ?? 0; // 0

const score = 0;
const totalScore1 = score ?? 100;
const totalScore2 = score || 100;

console.log(totalScore1);
console.log(totalScore2);
```

Workshop

- demo

Happy coding!