TypeScript

https://www.typescriptlang.org/docs/

JavaScript (JS) — динамічна, об'єктно-орієнтована[5] прототипна мова програмування. Реалізація стандарту ECMAScript.

JavaScript – є синхронною і однопотоковою (виконується по одній команді за раз)

Мова JavaScript використовується для:

- написання сценаріїв вебсторінок для надання їм інтерактивності
- створення односторінкових та прогресивних вебзастосунків (React, AngularJS, Vue.js)
- програмування на боці сервера (Node.js(Express.js))
- стаціонарних застосунків (Electron [Архівовано 1 серпня 2017 у Wayback Machine.], NW.js [Архівовано 16 червня 2020 у Wayback Machine.])
- мобільних застосунків (React Native [Архівовано 8 жовтня 2017 у Wayback Machine.], Cordova [Архівовано 14 червня 2021 у Wayback Machine.])
- сценаріїв в прикладних програмах (наприклад, в програмах зі складу Adobe Creative Suite чи Apache JMeter)
- всередині PDF-документів тощо.

https://uk.wikipedia.org/wiki/JavaScript

TypeScript - мова програмування, представлена Microsoft восени 2012; позиціонується як засіб розробки вебзастосунків, що розширює можливості JavaScript

· **Типізація**: TypeScript - це мова зі статичною типізацією. Типи змінних визначаються на етапі компіляції, що допомагає виявляти помилки до виконання програми.
· **Розширення JavaScript**: TypeScript додає типізацію та інші особливості до JavaScript. TypeScript код компілюється в JavaScript, який потім виконується в браузері або на сервері.
· **Інтерфейси та типи**: TypeScript підтримує інтерфейси та користувацькі типи, що дозволяє забезпечити структуру та безпеку коду.
· **Підтримка сучасних функцій**: TypeScript підтримує сучасні функції ECMAScript, такі як async/await, модулі, декоратори та багато іншого.

Переваги над JavaScript:

- можливість явного визначення типів (статична типізація),
- підтримка використання повноцінних класів (як у традиційних об'єктно-орієнтованих мовах),
- підтримка підключення модулів.

https://w3schoolsua.github.io/typescript/typescript_intro.html#gsc.tab=0

Типи даних

Тип даних є атрибутом даних, який містить інформацію для компілятора чи інтерпретатора,які програміст має намір використовувати дані.

Тип даних визначає:

- яку інформацію зберігає змінна (текст, число, логіку, структуру тощо)
- скільки пам’яті виділяється
- як представляється (кодується)
- які операції можна над нею виконувати
- як вона поводиться в пам’яті, при передачі, копіюванні або порівнянні

Типи даних

Тип даних є атрибутом даних, який містить інформацію для компілятора чи інтерпретатора,які програміст має намір використовувати дані.

У JS підтримуються :

- прості типи даних (відповідають скалярним значенням: числам, логічним значенням та ін.);
- складні типи даних посилання (об’єкти, функції).

Примітивні типи:
- **не є об’єктами**
- **не мають методів або властивостей**
- **зберігаються за значенням**, а не за посиланням

Примітивні (прості) типи:

1) **number** — цілі й дробові числа: 42, 3.14, -0.01, Infinity, NaN
   (64-бітне (8байтів) **число у форматі з плаваючою комою (IEEE 754)**)
2) **string** — текстові значення : 'hello', "React", `JWT`
3) **boolean** — логічне значення: true, false
4) **undefined** — коли змінна оголошена, але не має значення: let a; // a === undefined
5) **null** — явна "порожня адреса", тобто об'єкта не існує: const user = null
6) **symbol** — унікальні ідентифікатори (часто для приватних властивостей): const id = Symbol('unique')
7) **bigint** — для надвеликих чисел, що перевищують: Number.MAX_SAFE_INTEGER (запис містить «n» у кінці)
   Розмір = 16 байт (заголовок) + Math.ceil(кількість_бітів_у_числі / 64) * 8 байт (бітове. продставлення)
   const huge = 1234567890123456789012345678901234567890n

Для визначення типу даних може бути використана функцію `typeof`, яка повертає назву типу даних (рядок тексту)

```
typeof undefined // "undefined"

typeof 0 // "number"

typeof true // "boolean"

typeof "foo" // "string"

typeof function myFunc(){}   //"function"

typeof {} // "object"

typeof null // "object"

typeof  {name:'John', age:34} // "object"

typeof  [1,2,3,4]              // "object"
```

Значення	typeof
`42`	`'number'`
`'text'`	`'string'`
`true`	`'boolean'`
`undefined`	`'undefined'`
`null`	`'object'` ← особливість JS
`Symbol()`	`'symbol'`
`123n`	`'bigint'`
`{}`	`'object'`
`() => {}`	`'function'` ← спеціальний випадок

Опис змінних у Typescript

Загальна форма	Приклад
`let змінна: тип = значення;`	`let message: string = "Hello, TypeScript";`
`var змінна: тип = значення;`	`var count: number = 10;`

Опис констант у Typescript

Загальна форма	Приклад
`const змінна: тип = значення;`	`const pi: number = 3.14;`

Явно вказувати типи параметрів рекомендується тоді, коли:

- тип неочевидний або складний для виведення;
- потрібна додаткова документація чи обмеження.

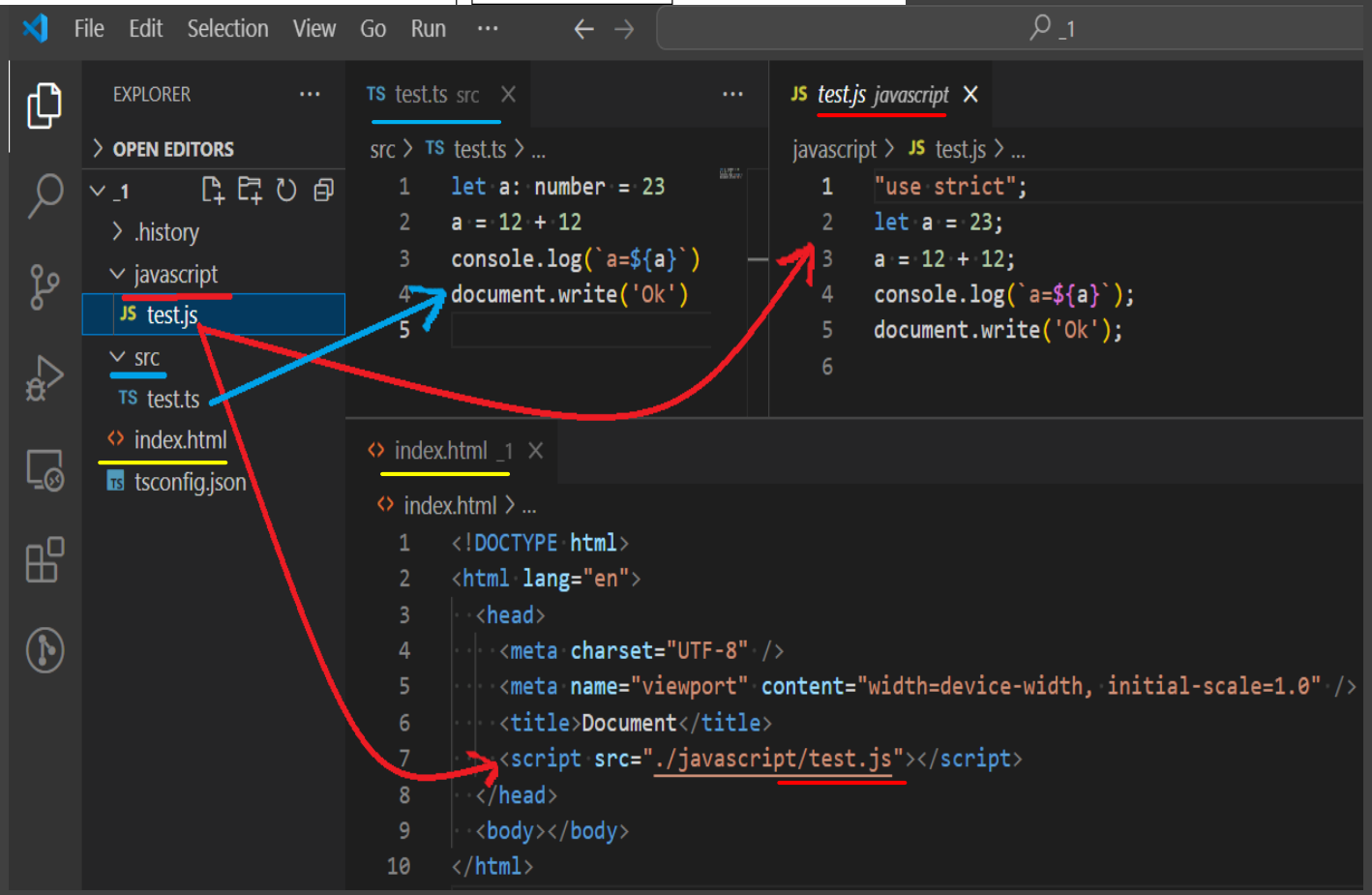
Загальна форма	Короткий приклад використання
`let varName: type;`	`let age: number = 30;`
`let name: string;`	`let userName: string = "Alice";`
`let isActive: boolean;`	`let isUserActive: boolean = true;`
`let bigNumber: bigint;`	`let largeValue: bigint = 9007199254740991n;`
`let uniqueId: symbol;`	`const id = Symbol('id'); let key: symbol = id;`

Використання TypeScript

Встановлення TypeScript:	`npm install -g typescript`

Ініціалізація TypeScript у проекті: (Перейдіть до директорії вашого проекту)	`tsc --init`
	
	//Ця команда створить файл `tsconfig.json`

Компіляція	//у терміналі виконуємо команду `tsc`
Автоматична компіляція після змін файлів «.ts»	//у терміналі виконуємо команду `tsc --watch`



Приклад tsconfig.json

```
{
 "compilerOptions": {
  "target": "es6",
  "module": "commonjs",
  "strict": true,
  "esModuleInterop": true,
  "forceConsistentCasingInFileNames": true,
  "outDir": "./javascript",
  "rootDir": "./src",
  "skipLibCheck": true
 },
 "include": [
  "src/**/*"
 ],
 "exclude": [
  "node_modules",
  " javascript "
 ]
}
```

- Файли створюємо у папці `src`
- Код компілюється у папку `javascript`
- У HTML файлі підключаємо скомпільовані файли «.js»

https://w3schoolsua.github.io/typescript/typescript_intro.html#gsc.tab=0

Ввести ім’я користувача і його вік. Якщо вік більше 17 привітати, інакше – повідомити про заборону відвідування сайту

`any`

- `any` - це тип, який відключає всі перевірки типів для змінної, дозволяючи їй приймати будь-яке значення та виконувати будь-які операції без помилок компіляції;
- використовується для швидкої інтеграції JavaScript-коду або коли тип даних дійсно невідомий.

```
let   змінна : any
```

```
let value: any

value = "hello";              // value  має значеняення типу string

value = 123;                  // value  має значеняення типу number
```

`unknown`

- `unknown` - це безпечніша альтернатива `any`, що також дозволяє змінній приймати будь-яке значення, але вимагає перевірки типу перед виконанням операцій.
- використовується, коли тип даних невідомий, але ви хочете зберегти безпеку типів, **змушуючи розробника явно перевіряти тип перед використанням.**

```
let змінна : unknown
```

```
let input: unknown = "test";


if (typeof input === 'string') {
    console.log(`String input: ${input.trim()}`);
  } else if (typeof input === 'number') {
    console.log(`Number input: ${input * 2}`);
  } else {
    console.log("Unknown input type.");
  }
}
```

Union-типи у TypeScript

Union (об’єднання) дозволяє вказати, що змінна, параметр чи результат функції може мати декілька можливих типів.

Загальна форма

```
let змінна: ТипA | ТипB | ТипC;
```

Приклади

```
let id: number | string;

id = 42;       // ✅ можна число
id = "user-1"; // ✅ можна рядок
```

```
let input: string | number | boolean;

input = "hello"; // ✅
input = 10;      // ✅
input = true;    // ✅
```

Користувач вводить ім’я. Якщо відмовиться, то заборонити доступ

Type Aliases (Псевдоніми типів)

Дозволяє створити нове ім’я для існуючого типу або комбінації типів

```
type ІмʼяТипу = Тип;
```

Тип може бути:

- Примітивним (string, number, boolean)
- Об’єктом ({ ... })
- Масивом (number[], Array<string>)
- Union-типом (string | number)
- Tuple ([number, string])
- Іншими псевдонімами або generics

Примітивні типи

```
type ID = string | number;

let userId: ID;
userId = 123;      // ✅
userId = "user1";  // ✅
userId = true;     // ❌ помилка
```

```
type Flag = boolean;

let isActive: Flag;
isActive = true;   // ✅
isActive = false;  // ✅
```

Об’єкт

```
type User = {
  id: number;
  name: string;
  email: string;
};

const u1: User = {
  id: 1,
  name: "Оля",
  email: "olya@example.com"
};
```

Масив

```
type Numbers = number[];
type Names = Array<string>;

let nums: Numbers = [1, 2, 3];
let names: Names = ["Іван", "Оля"];
```

Tuple

```
type Point = [number, string];

let p1: Point = [10, "X"];
```

Union

```
// Псевдонім для числа або булевого значення
type NumOrBool = number | boolean;

let flag: NumOrBool;
flag = 0;      // ✅
flag = true;   // ✅
flag = "no"; // ❌ помилка
```

Літеральні (злічені) типи TypeScript

Літеральний тип - це тип, який обмежує значення до конкретного літерального значення, а не лише до загального примітивного типу. Можемо навести повний список можливих значнь

```
let назва_змінної : значення1 | значення2 | значення3 | ... ;
```

Рядковий літеральний тип

```
let direction: "left" | "right";

direction = "left";    // ✅ дозволено
direction = "right";   // ✅ дозволено
direction = "up";      // ❌ помилка
```

Числовий літеральний тип

```
let statusCode: 200 | 404 | 500;

statusCode = 200;  // ✅
statusCode = 404;  // ✅
statusCode = 403;  // ❌ не входить у дозволені
```

Значення змішаного типу

```
let variableName: "дозволений_рядок" | 123 | true;

variableName = true   // ✅
variableName = false  // ❌ помилка

variableName = 123    // ✅
variableName = 17     // ❌ помилка


variableName = "дозволений_рядок"  // ✅
variableName = "інший_рядок"       // ❌ помилка
```

Об’єднання літеральних типів

```
type CustomType = "варіант1" | "варіант2" | "варіант3";
```

Приклади

```
type Direction = "left" | "right" | "up" | "down";

function move(dir: Direction) {
  console.log(`Рухаємося у напрямку: ${dir}`);
}

move("left");   // ✅
move("down");   // ✅
move("back");   // ❌ помилка
```

```
// Псевдонім для числа або літеральних значень
type NumOrBool = number | 'ok' | 'no'

let x: NumOrBool
x = 12    // ✅
x = 23    // ✅
x = 'ok'  // ✅
x = 'no'  // ✅
x= 'test' // ❌
```

```
type HttpMethod = "GET" | "POST" | "PUT" | "DELETE";

function request(url: string, method: HttpMethod) {
  console.log(`Запит: ${method} ${url}`);
}

request("/api/users", "GET");   // ✅
request("/api/users", "PATCH"); // ❌ помилка
```

Функції

Оголошення функції (Function Declaration)

```
function ім’яФункції(параметри: Тип): ТипПовернення {
  // тіло функції
}
```

```
function greet(name: string): string {
  return `Привіт, ${name}!`;
}

console.log(greet("Андрій"));
```

Функціональний вираз (Function Expression)

```
const ім’яФункції = function(параметри: Тип): ТипПовернення {
  // тіло
};
```

```
const greet = function(name: string): string {
  return `Привіт, ${name}!`;
};

console.log(greet("Оля")); // Привіт, Оля!
```

Стрілкова функція (Arrow Function)

```
const ім’яФункції = (параметри: Тип): ТипПовернення => {
  // тіло
};
```

```
const greet = (name: string): string => `Привіт, ${name}!`;
console.log(greet("Максим")); // Привіт, Максим!
```

Анотація типу функції

```
type MyFunc = (a: number, b: number) => number;

const add: MyFunc = (x, y) => x + y;
```

```
// Тип функції (контракт)
type Greeter = (name: string) => string;

// Реалізація, яка відповідає типу
const greet: Greeter = (name) => `Привіт, ${name}!`;

console.log(greet("Ірина")); // Привіт, Ірина!
```

З клавіатури вводиться номер місяця. За номером місяця визначити пору року

Перевизначення функцій у TypeScript

Загальна форма

```ts
// Сигнатури перевантаження
function fn(param: TypeA): ReturnTypeA;
function fn(param: TypeB): ReturnTypeB;

// Реалізація
function fn(param: TypeA | TypeB): ReturnTypeA | ReturnTypeB {
  // логіка
}
```

Приклад

```ts
// 1. Сигнатури
function double(x: number): number
function double(x: string): string

// 2. Реалізація
function double(x: number | string): number | string {
  if (typeof x === 'number') {
    return x * 2
  }
  return x + x
}

console.log(double(5)) // 10
console.log(double('Hi')) // "HiHi"
```

Скомільований

```ts
// 2. Реалізація
function double(x) {
    if (typeof x === 'number') {
        return x * 2;
    }
    return x + x;
}
```

Вказуємо можливі типи

```ts
// 1. Сигнатури
function sum(p1: number, p2: number): number
function sum(p1: string, p2: string): string
// 2. Реалізація
function sum(p1: number | string, p2: number | string): number | string {
  if (typeof p1 === 'number' && typeof p2 === 'number') {
    return p1 + p2
  }
  if (typeof p1 === 'string' && typeof p2 === 'string') {
    return parseInt(p1) + parseInt(p2)
  }
  throw new Error('Arguments must be both numbers or both strings')
}

document.write(sum(22, 11).toString()) //33
document.write(sum('9$', '9$'))        //18
```

Вказуємо any

```ts
// 1. Сигнатури
function sum(p1: number, p2: number): number
function sum(p1: string, p2: string): string
// 2. Реалізація
function sum(p1: any, p2: any): any {
  if (typeof p1 === 'number' && typeof p2 === 'number') {
    return p1 + p2
  }
  if (typeof p1 === 'string' && typeof p2 === 'string') {
    return parseInt(p1) + parseInt(p2)
  }
  throw new Error('Arguments must be both numbers or both strings')
}
```

Передається або номер дня, або назву дня на англійькій. Треба сказати чи вихідний чи робочий.

Масиви

Масиви в TypeScript - це впорядковані колекції значень одного типу або об'єднаних типів. Вони можуть бути визначені за допомогою синтаксису `Type[]` або `Array`.

Загальна форма

let arr: Type[];

```
let numbers: number[] = [10, 20, 30];
```

```
let names: string[] = ["Іван", "Оля", "Максим"];
```

```
let flags: boolean[] = [true, false, true];
```

let arr: Array<Type>;

```
let numbers: Array<number>;
numbers = [1, 2, 3, 4];
```

```
let names: Array<string>;
names = ["Іван", "Оля", "Максим"];
```

```
let flags: Array<boolean>;
flags = [true, false, true];
```

Планувальник відпусток. Дано масив імен водіїв і масив номерів, коли можнай йти у відпустку. Потрібно випадково вибрати ім’я водія і випадково вибрати номер місяця для відпустки.

Масиви

Union з масивами (перераховуємо можливі типи елементів)

let масив: (type1 | type2 | type3 | …) [ ]

Приклад

```
let mixed: (string | number)[] = [1, 'два', 3, 'чотири']

mixed.push(5) // ✅
mixed.push('шість') // ✅
mixed.push(true); // ❌
```

Сформувати масив значень ігрового барабана. Це або виграшна сума, або «Пауза», «Банкрот», «Супер приз»

Багатовимірні масиви

let ім’я_масиву : Type [ ] [ ] … [ ]

Двовимірний масив

```
let matrix: number[][];

matrix = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];
```

```
let matrix2: Array<Array<number>>;

matrix2 = [
  [10, 20],
  [30, 40]
];
```

Тривимірний масив

```
// Кожен елемент — двовимірний масив чисел
let cube: number[][][];

cube = [
  [
    [1, 2],
    [3, 4]
  ],
  [
    [5, 6],
    [7, 8]
  ]
];
```

Сформувати двовимірний масив (5*5) ігрового поля. Усі елементи нулі, або 1-корабель (4 штуки), або ‘block’- земля (5 штук)