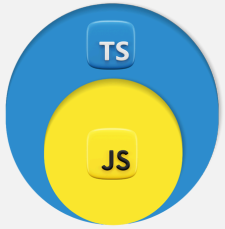
TypeScript Ściąga

Nauka Programowania i praca programisty witekpruchnicki.com

@witekpruchnicki

TypeScript to nadzbiór języka JavaScript, który dodaje statyczne typowanie, co pomaga wykrywać błędy już na etapie pisania kodu.

Rozszerza możliwości JavaScriptu, dodając funkcje jak interfejsy, typy i nowoczesne konstrukcje, a następnie kompiluje się do czystego JavaScriptu, dzięki czemu działa w dowolnym środowisku obsługującym JS.



Typy Podstawowe

TypeScript wprowadza typy takie jak number, string, boolean, array, tuple, enum, any, void, null, undefined, never, object.

```
let isDone: boolean = false;
let decimal: number = 6;
let color: string = "blue";
let empty: null = null;
let notAssigned: undefined = undefined;
function error(message: string): never {
  throw new Error(message);
}
let person: object = { name: "John", age: 30 };
```

Typy unii

Zmienna może być jednego z kilku typów.

```
let value: string | number;
value = "hello";
value = 42;
```

Generyki (Generics)

Tworzenie placeholderów, które mogą pracować z różnymi typami.

```
function identity<T>(arg: T): T {
  return arg;
}

let output1 = identity<string>("myString");
let output2 = identity<number>(100);
```

Enumy

Definiują zestaw nazwanych stałych.

```
enum Color {
  Red = 1,
  Green,
  Blue
}
let color: Color = Color.Green;
```

Any

Używane, gdy nie znamy typu zmiennej podczas pisania kodu.

```
let data: any = 42;
data = "now a string";
data = true;

Typ any pozwala na dużą elastyczność, ale warto go stosować
ostrożnie, by nie tracić kontroli nad typami w kodzie.
```

void

Wskazuje brak typu, najczęściej używany jako typ zwracany przez funkcje, które nic nie zwracają.

```
function warnUser(): void {
  console.log("This is my warning message");
}
```

Krotki (Tuples)

Tablice o ustalonej liczbie elementów z różnymi typami.

```
let person: [string, number];
person = ["Alice", 25];

let coordinates: [number, number, string];
coordinates = [10, 20, "N"];
```

Interfejsy

Definiują 'szablony' dla obiektów, klas i funkcji.

```
interface User {
  name: string;
  age: number;
}

const getUserInfo = (user: User): string => {
  return `User ${user.name} is ${user.age} years old.`;
};

// Przykład użycia
const user: User = { name: "Anna", age: 25 };
console.log(getUserInfo(user)); // "User Anna is 25 years old."
```

Funkcje

Typujemy parametry i wartości zwracane, obsługa parametrów opcjonalnych i domyślnych.

```
function buildName(firstName: string, lastName?: string) {
  if (lastName) return firstName + " " + lastName;
  else return firstName;
}

let result1 = buildName("Bob"); // OK
```

Klasy

Rozszerzenie klas ES6 o typowanie, modyfikatory dostępu, dziedziczenie.

```
class Greeter {
  greeting: string;

  constructor(message: string) {
    this.greeting = message;
  }

  greet(): string {
    return "Hello, " + this.greeting;
  }
}
```

Modyfikatory dostępu

public, private, protected kontrolują dostęp do metod i właściwości klasy.

```
class Animal {
  public name: string;    // dostępne wszędzie
  private age: number;    // dostępne tylko w tej klasie
  protected type: string; // dostępne w tej i klasach potomnych

  constructor(name: string, age: number, type: string) {
    this.name = name;
    this.age = age;
    this.type = type;
  }

  public getInfo(): string {
    return `${this.name} is a ${this.type}.`;
  }
}
```

Asercje typów

Pozwalają traktować zmienne jako inny typ.

```
let someValue: any = "this is a string";
let strLength: number = (someValue as string).length;
```

Tablice

Typujemy tablice jako typ[] lub Array<typ>.

```
// tablica liczb
let numbers: number[] = [1, 2, 3, 4, 5];
let numbersAlt: Array<number> = [1, 2, 3, 4, 5];
// tablica stringów
let words: string[] = ["hello", "world"];
let wordsAlt: Array<string> = ["hello", "world"];
//tablica mieszana
let mixed: (number | string)[] = [1, "two", 3, "four"];
let mixedAlt: Array<number | string> = [1, "two", 3, "four"];
```

Typy literałowe

Typy reprezentowane przez konkretne wartości, pozwalają na definiowanie zmiennych, które mogą przyjmować tylko określone wartości.

```
type Direction = "north" | "south" | "east" | "west";
let dir: Direction = "north";
```

Typ unknown

Bezpieczniejszy niż any, wymaga sprawdzenia typu przed użyciem.

```
let notSure: unknown = 4;

if (typeof notSure === "number") {
  let num: number = notSure;
}
```

Asercje typu Non-null

Używamy ! aby wskazać, że wartość nie jest null lub undefined, nawet jeśli kompilator uważa inaczej.

```
let name: string | null;
console.log(name!.length);
```

Operator opcjonalnego łańcucha

Bezpieczne odwołania do właściwości obiektów, które mogą być undefined lub null, używając ?.

```
let user = {
  name: "Bob",
  greet: () => "Hello!"
};

let greeting = user.greet?.(); // "Hello!"
```

Operator "Nullish Coalescing" (??)

Ustawia wartość domyślną, jeśli wartość jest null lub undefined:

```
let inputValue: string | null = null;
let result = inputValue ?? "Default Value"; //  "Default Value"
```

Type aliases (aliasy typów)

Pozwalają tworzyć niestandardowe nazwy dla złożonych typów, co ułatwia ich wielokrotne użycie i czytelność kodu.

```
type User = {
  name: string;
  age: number;
};

let newUser: User = { name: "Alice", age: 30 };
```

Używaj typowania jak najwięcej

Dodawanie typów do zmiennych, parametrów funkcji i zwracanych wartości pozwala wykryć błędy wcześniej i sprawia, że kod jest bardziej czytelny.

```
function add(a: number, b: number): number {
  return a + b;
}
```

Sprawdzaj istnienie wartości przed użyciem ich właściwości

Zamiast ryzykować błędy na undefined, używaj operatora opcjonalnego łańcucha (?.) do bezpiecznego dostępu do wartości.

```
let user = { name: "Alice", address: { city: "Wonderland" } };
console.log(user?.address?.city);

// jeśli address jest undefined, nie będzie błędu
```

Typy przecięcia

Łączą wiele typów w jeden, używamy & do łączenia.

```
interface A {
  a: string;
}
interface B {
  b: number;
}

type C = A & B;

let obj: C = { a: "test", b: 123 };
```

Unikaj użycia 'as'

Używaj rzutowania typów (as) ostrożnie – jeśli TypeScript podpowiada błąd typów, to najczęściej jest to słuszne ostrzeżenie, a nie coś, co powinno być przez Ciebie ignorowane.

Poznaj różnicę między null, undefined, a void

null oznacza brak wartości, undefined jest wartością domyślną niezainicjowanych zmiennych, a void jest używany w funkcjach, które nic nie zwracają.

Korzystaj z 'any' z rozwagą

Typ any może czasem być pomocny, ale gdy go używasz to tracisz korzyści wynikające z typowania w TypeScripcie. Używaj go tylko wtedy, gdy naprawdę nie znasz typu lub pracujesz z bardzo dynamicznymi danymi.

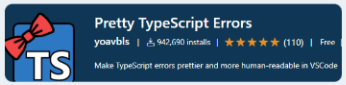
Słowa na start

To, że TypeScript na początku będzie Cię denerować to normalne. Nie przejmuj się tym, bo każdy tak ma jeśli przechodzi z JS-a do TS-a.🙂

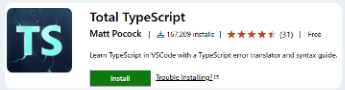
Wchodzisz w świat, gdzie błędy są wyłapywane wcześniej, a Twój kod ma być bardziej czytelny. Sam sobie za to podziękujesz, kiedy wrócisz do niego za jakiś czas 😉

Działaj krok po kroku, baw się nauką, a zobaczysz, jak wszystko zacznie się układać 💪👩‍💻

Narzędzia i przydatne linki



Formatuje i koloruje komunikaty błędów TypeScript, dzięki czemu stają się bardziej czytelne i zrozumiałe w konsoli, co ułatwia szybkie rozwiązywanie problemów w kodzie.



Total TypeScript to rozszerzenie do Visual Studio Code, które ułatwia naukę TypeScriptu bezpośrednio w edytorze. Dzięki niemu zyskasz pomocne podpowiedzi dotyczące składni oraz przetłumaczenie złożonych komunikatów błędów TypeScript na prosty, zrozumiały język.

Tutoriale w formie tekstu

https://www.typescriptlang.org/docs/handbook/intro.html
https://www.typescripttutorial.net/
https://www.tutorialsteacher.com/typescript
https://codemastery.dev/ts/interactive-handbook/intro/0
https://type-level-typescript.com/