Understanding TypeScript - 2020 Edition

# Section 1: Getting Started

## 01. Welcome to the Course!

## 02. Why Is TypeScript & Why Should You Use It?

## 3. Installing & Using TypeScript

### ! as HTMLInputElement;

const input1 = document.getElementById("num1")! as HTMLInputElement;

const input2 = document.getElementById("num2")! as HTMLInputElement;

inseamna ca nu este null si este un HTML input element!

button.addEventListener("click", function() {

  console.log(add(+input1.value, +input2.value));

  const result = add(+input1.value, +input2.value)

  document.getElementById('result').innerHTML = result;

});

### +input1.value

**+ transforma string to number**

## 4. TypeScript Advantages – Overview

## 5. Course Outline

## 6. How To Get The Most Out Of The Course

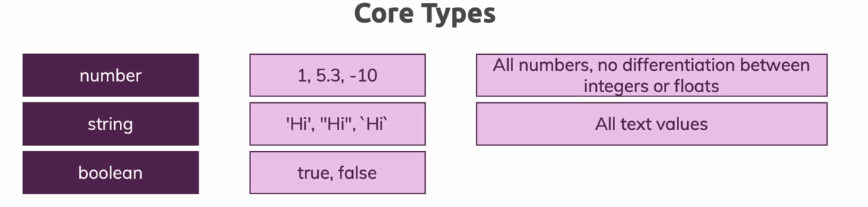
## 7. Setting Up A Code Editor / IDE

## 8. The Course Project Setup

# Section 2: TypeScript Basics & Basic Types

## 9. Module Introduction

## 10. Using Types



function add(n1, n2) {

  return n1 + n2

}

const number1 = 5;

const number2 = 2.8;

const result = add(number1, number2)

console.log(result); //7.8

const number1 = '5';

console.log(result); //52.8

**Error**

**app.ts:8:20 - error TS2345: Argument of type '"5"' is not assignable to parameter of type 'number'.**

**8 const result = add(number1, number2)**

## 11. TypeScript Types vs JavaScript Types

**Ptr. js**

function add(n1:number, n2:number) {

if(typeof n1 !== 'number' || typeof n2 !== 'number') {

  throw new Error('Incorrect input!');

}

  return n1 + n2

}

const number1 = '5';

const number2 = 2.8;

const result = add(number1, number2)

console.log(result); //7.8

app.js:3 Uncaught Error: Incorrect input!

at add (app.js:3)

at app.js:9

add @ app.js:3

(anonymous) @ app.js:9

## 12. Important: Type Casing

### string

### number

In TypeScript, you work with types like string or number.

**Important**: It is string and number (etc.), **NOT** String, Number etc.

**The core primitive types in TypeScript are all lowercase!**

## 13. Working with Numbers, Strings & Booleans

function add(n1:number, n2:number, showResult: boolean, phrase:string) {

  const result = n1 + n2

  if(showResult) {

    console.log(phrase + result)

  } else {

    return result

  }

}

const number1 = 5;

const number2 = 2.8;

const printResult = true;

const resultPhrase = 'Result is: '

const result = add(number1, number2, printResult, resultPhrase )

## 14. Type Assignment & Type Inference

### Type Inference

Pentru aceste variabile nu trebuie as mai declaram type-ul! Deoarece ar fi redundant!

const number1 = 5;

const number2 = 2.8;

const printResult = true;

const resultPhrase = 'Result is: '

insa…

Daca avem acest caz:

//caz

let number1;

number1 = '5';

trebuie sa declaram tipul variabile number1

**correct!!**

//caz

let number1:number;

number1 = 5;

## Quiz 1: Understanding Types

**Why are "Types" useful and offer an advantage compare to vanilla JavaScript?**

Good job!

That's correct! Types allow you to detect if data of a wrong type is used or pass around, hence you can avoid unexpected runtime errors.

**Will the following code throw a compilation error?**

1. **let userName: string;**
2. **userName = 'Maximilian';**
3. **userName = false;**

yes, assigning a boolean to a variable which was assigned a "string" types is not allowed and will yield a compilation error.

**Does this code rely on type inference?**

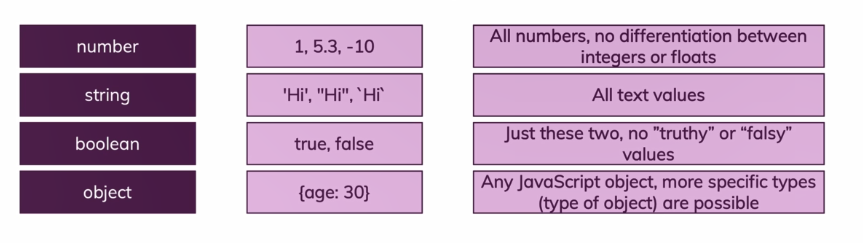
1. **const age: number = 29;**

 TypeScript would be able to infer the type (and hence you should omit ":number") but here, we actually also have an explicit type assignment.

**What's the difference between JavaScript types (e.g. typeof 'Max' => 'string') and TypeScript types (e.g. const name: string = '...')?**

 JS has no compilation step but at runtime, you can check for certain types (e.g. in if conditions). TS on the other hand allows you to catch certain errors during development since it checks types during compilation as well.

## 15. Object Types



const person:{

  name:string;

  age:number;

} = {

  name: 'Max',

  age: 30

};

console.log(person.name)

## 16. Nested Objects & Types

Of course object types can also be created for**nested objects**.

Let's say you have this JavaScript **object**:

const product = {

  id: 'abc1',

  price: 12.99,

  tags: ['great-offer', 'hot-and-new'],

  details: {

    title: 'Red Carpet',

    description: 'A great carpet - almost brand-new!'

  }

}

This would be the **type** of such an object:

{

  id: string;

  price: number;

  tags: string[],

  details: {

    title: string;

    description: string;

  }

}

So you have an object type in an object type so to say.

Pentru a functiona asta trebie pusa in **interface!**

### Interface

interface person {

  name:string;

  age:number;

}

const person:person = {

  name: 'Max',

  age: 30

};

console.log(person.name)

interface product {

  id: string;

  price: number;

  tags: string[],

  details: {

    title: string;

    description: string;

  }

}

const product = {

  id: 'abc1',

  price: 12.99,

  tags: ['great-offer', 'hot-and-new'],

  details: {

    title: 'Red Carpet',

    description: 'A great carpet - almost brand-new!'

  }

}

console.log(product.id)

## 17. Arrays Types



//17. Arrays |Types

const member = {

  name: 'Max',

  age: 30,

  hobbies: ['Sports', 'Cooking']

};

let favoriteActivities: string[];

favoriteActivities = ['sport']

**Error**

favoriteActivities = ['sport', 5]

for (const hobby of member.hobbies) {

  console.log('hobby:', hobby);

}



## 18. Working with Tuples



//18.Tuples

member.role.push('admin');

member.role[1] = 10;

for (const r of member.role) {

  console.log('role:', r);

}

const member: {

  name: string,

  age: number,

  hobbies: string[],

  role:[number, string]

} = {

  name: 'Max',

  age: 30,

  hobbies: ['Sports', 'Cooking'],

  //tuples

  role: [2, 'author']

};

### Tuples

Ptr tuples folosim

  role:[number, string]

/18.Tuples

//an array with 2 values

//member.role.push('admin');

 member.role[0] = 10;

for (const r of member.role) {

  console.log('role:', r);

}

console.log(member.role)

## 19. Working with Enums

### enums

//19.Enums

const ADMIN = 0;

const READ\_ONLY = 1;

const AUTHOR  = 2;

const person2 = {

  name: 'Max',

  age: 30,

  hobbies: ['Sports', 'Cooking'],

  role:ADMIN

};

if(person2.role === ADMIN) {

  console.log('is admin') //is admin

 }

Constantele le punem intr-un enum

enum Role {ADMIN, READ\_ONLY, AUTHOR}

si folosim ca un obiect!!

const person2 = {

  name: 'Max',

  age: 30,

  hobbies: ['Sports', 'Cooking'],

  role: Role.ADMIN

};

if(person2.role === Role.ADMIN) {

  console.log('is admin') //is admin

 }

enum Role {ADMIN = 'ADMIN' , READ\_ONLY = 100, AUTHOR = '200'}

## 20. The "any" Type

let favoriteActivities2: any[];

favoriteActivities2 = ['sport', 5 , [2, 5]]

## 21. Union Types

function combine(n1:number, n2:number) {

  const result = n1 + n2

    return result;

}

const combinedAges =  combine(30,26);

console.log('combinedAges:', combinedAges);

Asa nu va functiona

const combinedNames =  combine('Max', 'Gigi');

refactorizam

function combine(n1:number | string, n2:number | string) {

  let result;

  if(typeof n1 === 'number' && typeof n2 === 'number') {

    result = n1 + n2

  } else {

    result = n1.toString() + n2.toString();

  }

  //!!

  return result;

}

const combinedAges =  combine(30,26);

console.log('combinedAges:', combinedAges);

const combinedNames =  combine('Max', 'Gigi');

console.log('combinedNames:', combinedNames);

## 22. Literal Types

function combine(n1:number | string, n2:number | string, resultType:'as-number' | 'as-text') {

  let result;

  if(typeof n1 === 'number' && typeof n2 === 'number' || resultType === 'as-number' ) {

    result = +n1 + +n2

  } else {

    result = n1.toString() + n2.toString();

  }

  //!!

  return result;

}

const combinedAges =  combine(30,26, 'as-number');

console.log('combinedAges:', combinedAges);

const combinedStringAges =  combine('30', '26', 'as-number');

console.log(combinedStringAges)

const combinedNames =  combine('Max', 'Gigi', 'as-text');

console.log('combinedNames:', combinedNames);

## 23. Type Aliases / Custom Types