Corso TypeScript

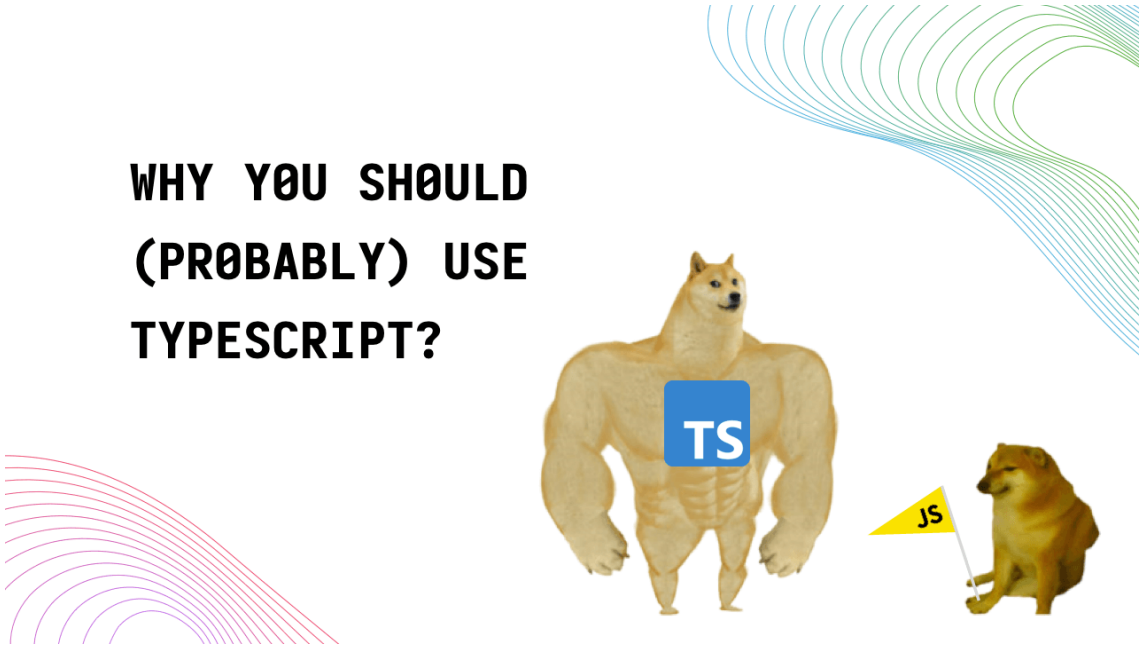


Figura 1 Corso: https://www.youtube.com/watch?v=BwuLxPH8IDs

Capitolo 1 - Introduzione

Cosa è TS?

TS è un **superset** di JavaScript che permette di definire con maggiore rigore il **tipo di dato**

Javascript, essendo **debolmente tipizzato**, incorre in degli errori e in situazioni di **coercion** che pongono ad artefatti di logica ("1" + 1 = 11).

Con TS questa cosa viene **superata**!

Gli errori si presenteranno in fase di **transpiling** e non in **runtime**.

Non essendo **leggibile** dagli environment adatti a JS, come i **browser o Node.js**, TS deve essere transiplato in plain JS (tramite Babel) per essere poi eseguito.

JavaScript usa una **tipizzazione dinamica** e **debole** (usata in **runtime**), **TypeScript** una **statica** e **forte** (usata in **development**)

Capitolo 2 – Installazione

1) Installare Node.js
2) Installare typescript: **npm install -g typescript**
3) Controlla la versione tsc -v (N.B. : **tsc.cmd** -v permetterà di aggirare il blocco delle policies del SO)
4) Crea un file in .ts di test
5) Compilare il file di test in plain js → **tsc(.cmd) nome_file.ext**
6) Linkare il file js corretto per eseguire lo script appena transpilato

N.B.: Typecasting

```
const input1 = document.getElementById("num1")
```

// Tipo di input in html

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

// Aggiunta del typecasting, ovvero uno "specificatore tipico di ts che specifica il tipo di elemento htm

```
function add(num1: number, num2: number) {
```

// Specificare il tipo è il core di ts

N.B. La presenza di un altro file js contestuale verrà considerato come un errore da ts.

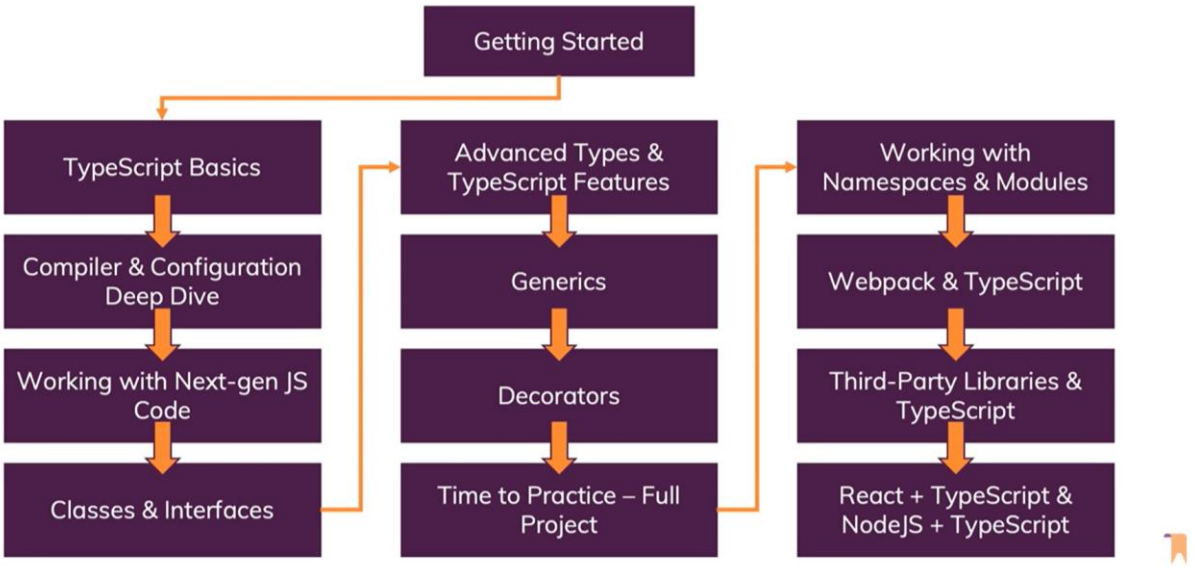
Capitolo 3 – Perché scegliere TS

Quali sono i pro di TS?

- Specifica i datatype
- Permette l'uso di features più moderne di js, poi transpilate
- Uso di **Generics*** ed **Interfaces***
- Aggiunta dei **Decorators***
- Alta configurabilità
- Presenza di strumenti adatti alla facilitazione dello sviluppo

*In particolare questi danno un aiuto solo in **fase di sviluppo**.

Capitolo 4 – Panoramica di TS



Capitolo 5 – I tipi di dato

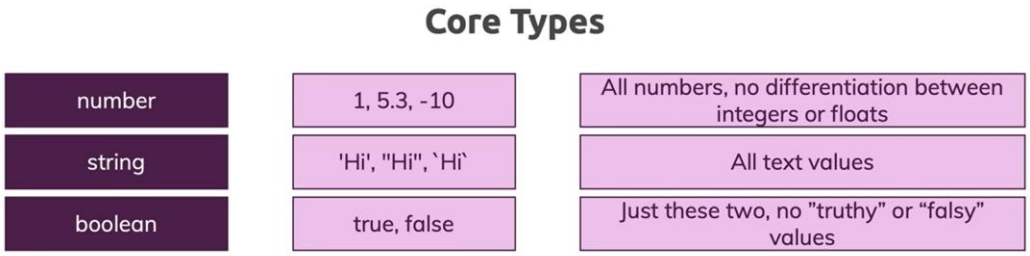


Figura 2 "Comparazione tra tipi di dato in JS (sinistra) e in TS (destra)"

Esempio di warning di errore in caso di tipo di dato dichiarato errato

```
function add(n1: number, n2: number) {
    return n1 + n2;
}
const number1 = "5";
const number2 = 2.8;
const result = add(number1, number2);
console.log(result);
```

// Codice

```
app.ts:8:20 - error TS2345: Argument of type 'string' is not assignable to parameter of type 'number'.

const result = add(number1, number2);

                ~~~~~~~

Found 1 error in app.ts:8
```

// Errore in terminale

N.B. : TS non blocca la compilazione, segnala soltanto l'errore!

```
function add(n1: number, n2: number) {
    return n1 + n2;
}
const number1 = 5;
const number2 = 2.8;
const result = add(number1, number2);
console.log(result);
```

// Errore riparato

N.B.: in JS sarebbe servito un throw dell'errore che avrebbe reagito solo in runtime

```
/*     if (typeof number1 !== 'number' || typeof number2 !== 'number' ) {
        throw new Error('Input Errato!')
    } */
```

Capitolo 6 – Numeri, stringhe e booleani

Esempio di **dichiarazione dei tipi** negli arguments di una funzione:

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

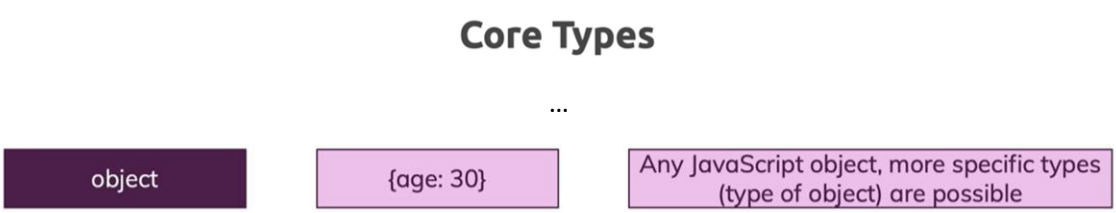
…

Type Inference (Deduzione del tipo): TS è capace di intuire il tipo della variabile che gli si presenta davanti, anche se questo non viene esplicitato ad ogni dichiarazione di essa.

```
let number1= 5;
number1 = "5";
```

// L'assegnazione di un nuovo tipo segnalerà un errore

Capitolo 7 – Oggetti



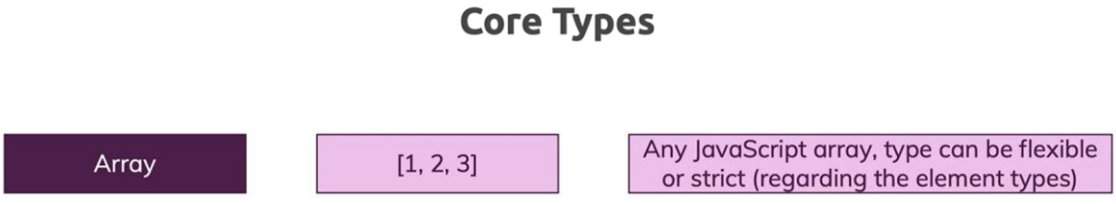

```
const person: object = {
    name: 'Maximilian',
    age:30
};
```

// Come definire il tipo “oggetto”, ma in modo **aspecifico**.

```
const person: {
    name: string;
    age: number;
} = {
    name: 'Maximilian',
    age:30
};
```

// Come definire il tipo “oggetto”, ma in modo **specifico.**

Capitolo 8 – Array



Nella determinazione di un array bisogna definirne anche il contenuto.

Es.

let arr = **string[]**;

arr = [“string1”, “string2”, …];

L’inserimento di **item con tipo errato** porterà ad un **errore**

La creazione di un **array** con **tipo misto** dovrà essere definita con **any[]**.

Per sillogismo saranno utilizzabili solo i metodi adatti al **tipo di dato** definito per **quell'array specifico**

Es:

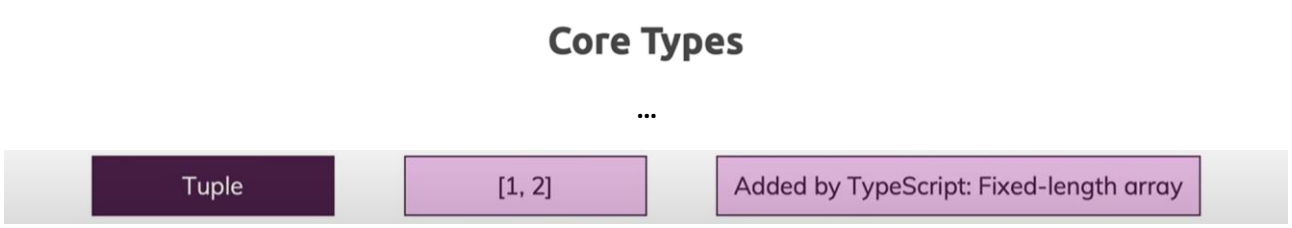
…

```
    age: number;
    hobbies: string[];
} = {
    name: 'Maximilian',
    age:30,
    hobbies: ['Sports', 'Cookies']
};


for (const hobby of person.hobbies) {
    console.log(hobby.toUpperCase())
    // console.log(hobby.map) --> Darà un ERRORE
}
```

…

Capitolo 9 – Tuple



Cosa sono:

I Tuples sono array con una lunghezza fissa del numero di items.

JavaScript non presenta questa tipologia di dato.

Es.:

```
var employee: [number, string] = [1, "Steve"];

var person: [number, string, boolean] = [1, "Steve", true];
```

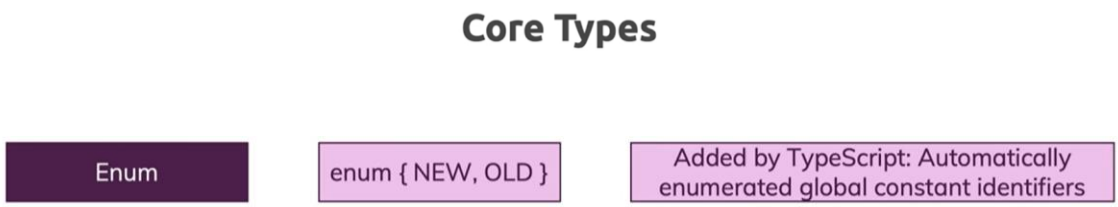
// Esempio di tuples, arrays con rispettivamente 2 elementi e 3 elementi

Es. 2:

```
const person: {
    name: string;
    age: number;
    hobbies: string[];
    role: [number, string];
} = {
    name: 'Maximilian',
    age:30,
    hobbies: ['Sports', 'Cookies'],
    role: [1, 'author']
};
```
// Creazione di un modello corretto in un oggetto

```
person.role[0] = 'string'
```
// L'inserimento di un tipo non corretto o fuori lunghezza massima, comporterà un **errore**

Capitolo 10 – Enum



…

Cosa sono:

Gli **enum** sono strutture dati, non presenti in JS, che consistono in enumerazioni di tipo di dato.

Possono essere **numerici**, in **stringhe** o **eterogenei**

Es. **Enum numerico**

```
enum PrintMedia {
  Newspaper,
  Newsletter,
  Magazine,
  Book
```

```
Newspaper = 1
Newsletter = 0
Magazine = 3
Book = 2
```

PrintMedia.Newspaper // 1 - PrintMedia[Magazine] // 3

Capitolo 11 – Any

Core Types

…

Any	*	Any kind of value, no specific type assignment

Cosa sono?

Any permette di accettare qualsiasi tipo di dato!

Es.:

let variables = any[]

variables = [“str”, 1, true, 3, “str2”, obj]

Capitolo 12 – Union type

Cosa sono?

Gli Union type sono i risultati di unioni di tipi di dati che danno maggiore flessibilità alla variabile definita

Es.:

```
function combine(input1: number | string , input2: number | string) {
```

…

Capitolo 13 – Literal type

Cosa sono?

I literal type servono ad indicare non soltanto il tipo di dato, ma il valore “specifico” che quella variabile deve avere.

Es.:

```
      resultConversion: string =/= resultConversion: 'as-string' | 'as-number'
```

Esempio esteso:

…

```
    input2: number | string,
    resultConversion: 'as-string' | 'as-number'
    ) {

    let result;

    if (typeof input1 === 'number' && typeof input2 === 'number' ||
resultConversion === 'as-number') {
        result = +input1 + +input2;
```

…

Capitolo 14 – Aliases Type

(type nome_combinazione)

Gli **Aliases** permettono di incamerare union types (o anche solo dichiarazione di tipo di dato) in un'unica variabile di consumo da usare in base alle eseigenze

Es.:

```
type Combinable = number | string;
```

// Una parola chiave che incamera in Combinable l’union type

```
type Combinable = number | string;

function combine(
```

```
    input1: Combinable ,
    input2: number | string,
```

Capitolo 15 – Funzioni avanzate

E' possibile definire con precisione anche il tipo di dato che il **return** di una funziona dovrebbe avere come output

Es.:

… : **number** { …

```
function add(num1: number, num2: number) : number {
    return num1 + num2;
}
// …ma è meglio evitare di specificarlo, a meno che strettamente necessario…
```

E' possibile inserire varie tipologie di dato come output di una funzione:

N.B. **Void** è un parametro inserito in automatico da TS, come nel caso del number sopra, per specificare l'assenza di un **return** nella funzione d'uso.

Es.:

…

```
function printResult(num: number): void {
    console.log('Result is: ' + num)
}
printResult(add(1,2));
// void è implicito, ma può essere esplicitato, come in java.
```

------ Caso Limite ------

N.B. **Undefined** è un tipo di dato utilizzabile in TypeScript.

Es.:

```
function printResult(num: number): undefined {
    console.log('Result is: ' + num)
```

```
}
printResult(add(1,2));
```

// Lo specificarlo farà incorrere in un errore, poiché una funzione **non dovrebbe** restituire un **undefined**.
// L'errore terminerà se si inserirà un **return**.

------ Caso Limite ------

E' possibile specificare anche che una variabile sia una funzione.

Questa è poi possibile uguagliarla ad una funzione già esistente.

Esempio:

Funzione 1

```
function add(num1: number, num2: number) : number {
    return num1 + num2;
}
```

Variabile definita con il datatype Function

```
let combinable: Function;
```

Uguaglianza

```
combinable = add;
```

N.B.: Una **definizione** più **precisa** della **funzione** è possibile effettuarla con un **arrow function**

Es.:

let variabile = () => number

// Deve essere una funzione senza parametri (args) che dovrà restituire un numero (return).

Es.2:

let variabile = (a: number, b: number) => number

// Deve essere una funzione con 2 parametri (args) di tipo numero che dovrà restituire un numero (return).

Esempio di definizione di una callback:

```
// Callback
function addAndHandle(num1: number, num2: number, cb: ( num:number) => void) {
    let result = num1 + num2;
    cb(result);
}
```

```
addAndHandle(2,4, (result) => {
    console.log(result)
});
```

Capitolo 16 – Unknown Type

Figura 3 Unknown o Unown?

Cosa è?

Unknown è un tipo indefinito (diverso da undefined) il cui tipo di dato è ancora **sconosciuto**.

Esso potrà essere modificato con il tipo di dato che si vuole, senza che ci siano errori in fase di sviluppo.

Di **default** il tipo di dato di una variabile è **any**.

Es.:

```
let userInput: unknown;
userInput = 5;
userInput = 'string';
```
// Non darà errore

Una **comparazione tra tipi** farà capire che il tipo unknown è differente:

```
let userInput: unknown;
let userName: string;
userInput = userName // Ok
userName = userInput; // Errore
```

Capitolo 17 – Never Type

Cosa è?

Il tipo **never** permette di effettuare un **return** di una funziona, **senza doverlo esplicitare** e senza avere un risultato **undefined** che potrebbe definire degli errori in runtime.

Es.

```
function generateError( message: string, code: number) : never {
    throw {message: message, erroCode: code};
}
generateError(`C'è stato un errore`, 500);
// Codice
```

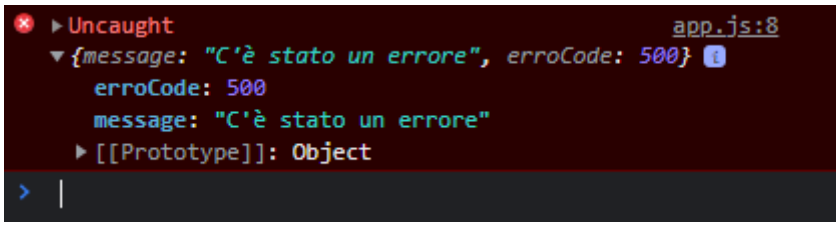


Figura 4 La funzione non ha un return esplicito, ma non restituisce un undefined

Capitolo 18 – Interfaces*

***Integrazione da corso**: https://www.youtube.com/watch?v=gp5H0Vw39y

Cosa sono?

Le **Interfaces** sono strutture che definiscono una mappatura dell'oggetto o della struttura dati su cui sono usate.

Es.

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

Oggetto che usa una interface:

```
const user: User = {
    name: 'Jack',
    age: 30
}
```

Oggetto che non usa una interface, ma ha una struttura uguale:

```
const user2: {
    name: string,
    age: number
} = {
    name: 'Paul'
}
```

// Sarà presente un errore perché age non è definito

N.B. : La struttura dell'Interface è **mandatoria**, ma è possibile rendere opzionali certe proprietà usando '**?**'

Es.:

```
age?: number
```

Es.: Integrazione di funzioni in inteface

```
interface User {
    name: string,
    age?: number,
    getMessage(): string
}

const user: User = {
    name: 'Jack',
    age: 30,
    getMessage() {
        return "Hello " + this.name
    }
}
```

// Hello Jack

Capitolo 19 – Uso del DOM*

***Integrazione da corso**: https://www.youtube.com/watch?v=gp5H0Vw39y

Nell'**uso del DOM** TS fornisce degli strumenti molto utili che permettono di definire la tipologi dell'elemento in uso, su cui, ad esempio, è possibile apporre un evnto.

Es. :

```
const someElement = document.querySelector('.foo');

someElement.addEventListener('blur', (event) => {
    const target = event.target as HTMLInputElement;
    console.log(target.value);
})
```

// as HTMLInputElement → Specificazione che l'elemento è un input HTML

Capitolo 20 – Classi*

***Integrazione da corso**: https://www.youtube.com/watch?v=gp5H0Vw39y

Es. :

```
class User {
    firstName: string;
    lastName: string;

    constructor(firstName: string, lastName: string) {
        this.firstName = firstName,
        this.lastName = lastName
    }

    getFullName(): string {
        return 'Hello' + ' ' + this.firstName + ' ' + this.lastName;
    }
}

const user = new User('Jack', 'Giuliani');

console.log(user.getFullName())
```

public e private:

2 keyword che permettono di specificare che le proprità definite nella classe siano utilizzabili solo dentro di essa o anche al di fuori

```
public firstName: string;
private lastName: string;
```

// il lastName darà errore se consoleloggato (o genericamente usato fuori), mentre il firstName no.

readonly:

Key word che rende una proprietà di sola lettura

static:

Key word che definisce una proprietà statica, ovvero non mutabile

---

Capitolo 21 – Generics*

Generics usa la variabile di tipo <T>, un tipo speciale di variabile che denota i tipi.

La variabile ricorda il tipo fornito dall'utente e lavora solo con quel particolare tipo.

Questo è chiamato conservazione delle informazioni sul tipo.

Es.:

```
function getArray<T>(items : T[] ) : T[] {
    return new Array<T>().concat(items);
}
let myNumArr = getArray<number>([100, 200, 300]);
let myStrArr = getArray<string>(["Hello", "World"]);
```

Capitolo XX – Utilities tools

- **Watch Node:**

Permette la **ricompilazione automatica** ad ogni **cambiamento** effettuato sul **file .ts**

```
tsc.cmd app.js --watch
```

- **Compilazione intera cartella progetto:**

```
tsc.cmd --init
```

// Creerà un file di configurazione

```
tsc.cmd
```

// Transpilerà tutti ifiles situati nella stessa cartella del file di configurazione

N.B. : E' utilizzabile anche con **watch node** (**--watch**).

- **Moduli e configurazione:**

…

```
    "skipLibCheck": true /* Skip type checking all .d.ts files. */
  },
  "exclude": [
    "basics.ts", // Esclude il file basics.ts
    "**/*.ts" // Esclude tutti i file con questa estensione con questo path
  ],
  "include": [
    "app.ts"
  ]
}
```

// Inserimento dei files da **escludere/includere** dalla transpilazione

N.B.: E' possibile inserire anche intere cartelle

- **Target Compiling:**

```
"target": "es2016" /* Set the JavaScript language version for emitted
JavaScript and include compatible library declarations. */
```

// Specifica in che vers. di JavaScript verranno transpilati i files

- **Definizione di una cartella di input ed una di output per la transpilazione**

```
{
  "compilerOptions": {
    "rootDir": "./src",
    "outDir": "./dist",
```

…

Risorsa aggiuntiva:

https://www.tutorialsteacher.com/typescript
