



Typescript

Introdução, declaração de variáveis e tipos básicos de dados

Ely – ely.miranda@ifpi.edu.br

1

TypeScript



- É um super conjunto (super set) de JavaScript;
- Adiciona tipagem estática e explícita;
- Site oficial: <https://www.typescriptlang.org/>

ely.miranda@ifpi.edu.br

2

2

TypeScript



- Open source e desenvolvida pela Microsoft;
 - Mesmo criador do C# e Delphi;
- Larga adoção sendo utilizado com muitos frameworks conhecidos como Angular, React, Express...

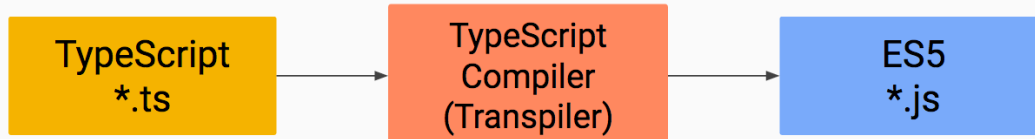
ely.miranda@ifpi.edu.br

3

3

No navegador

- O navegador não entende Typescript;
- O código é “transpilado” para JavaScript.

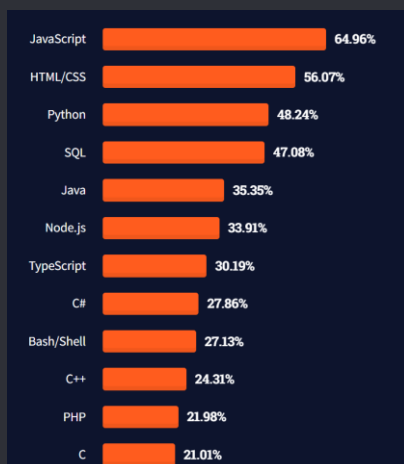


ely.miranda@ifpi.edu.br

4

4

Por que (o ecossistema) JavaScript?



ely.miranda@ifpi.edu.br

5

5

Por que não apenas JavaScript?

- A tipagem dinâmica do JavaScript leva a erros de runtime:

```
function soma(a, b) {  
    return a + b;  
}
```

```
console.log(soma(1,2));  
console.log(soma(1,"2"));  
console.log(soma(1,true));  
console.log(soma(1,null));
```

ely.miranda@ifpi.edu.br

6

6

Por que não apenas JavaScript?



- A tipagem dinâmica do JavaScript leva a erros de runtime:

```
function soma(a, b) {
    return a + b;
}

console.log(soma(1,2)); //3
console.log(soma(1,"2")); //12
console.log(soma(1,true)); //2
console.log(soma(1,null)); //1
```

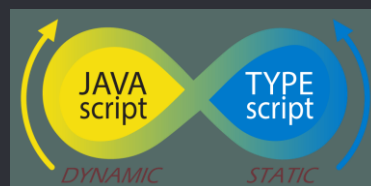
ely.miranda@ifpi.edu.br

7

7

Por que não apenas JavaScript?

- Tipagem estática do TypeScript leva à confiabilidade de software;
- Menos erros ocorrem em tempo de execução:
 - Com a exposição de *warnings* e erros durante o projeto/ desenvolvimento;
 - Menos erros “descobertos” por usuários.



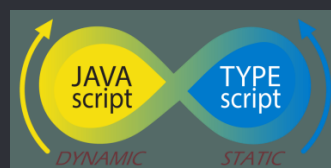
ely.miranda@ifpi.edu.br

8

8

Por que não apenas JavaScript?

- Com tipagem dinâmica e implícita:
 - Pouco se pode inferir sobre as construções em tempo de projeto;
 - Pouco se pode avançar em ferramentas como: “Linters”, Typecheckers e Intellisenses.



ely.miranda@ifpi.edu.br


9

9

Por que não apenas JavaScript?

- Temos também mais produtividade:
 - Menos tempo tentando resolver bugs.

Why TypeScript?



We love TypeScript for many things... With TypeScript, several of our team members have said things like 'I **now actually** understand most of our own code!' because they can easily traverse it and understand relationships much better. And we've found several bugs via TypeScript's checks."

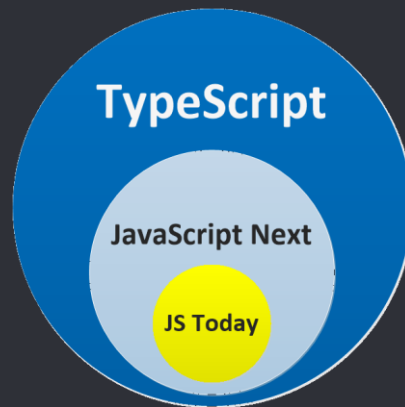
— Brad Green, Engineering Director - AngularJS

ely.miranda@ifpi.edu.br

10

10

TypeScript



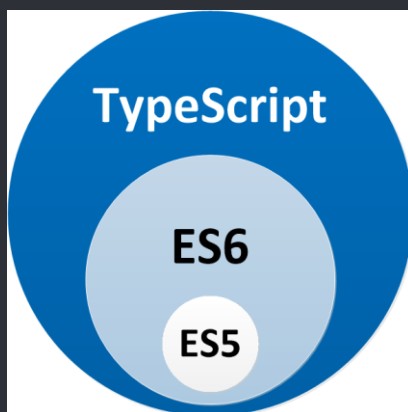
https://topnyjets.com/update_typescript_javascript_diagram.php

ely.miranda@ifpi.edu.br

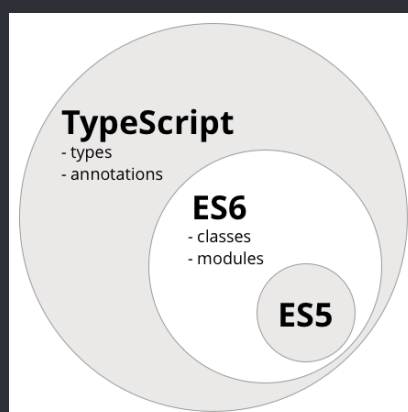
11

11

TypeScript



<https://medium.com/@basarat/typescript-won-a4e0dfe4b08>



https://topnyjets.com/update_typescript_javascript_diagram.php

ely.miranda@ifpi.edu.br

12

12

Requisitos de software

- Node.js;
- VS Code;
- TypeScript:
 - `npm install typescript -g`
 - `tsc -v`



ely.miranda@ifpi.edu.br

13

13

Primeiro programa

- Crie um arquivo chamado `index.ts` no VS Code e insira as duas instruções:

```
const myName: string = "Ely";  
console.log("Hello, " + myName);
```

- No prompts de comando execute:

```
tsc index.ts  
node index.js
```

ely.miranda@ifpi.edu.br

14

14

Sobre a “transpilação”

- O código que executamos ao final é JavaScript nativo transpilado;
- É importante observar o código gerado em uma fase inicial de estudos;
- O arquivo gerado é sobrescrito a cada nova transpilação com o comando `tsc`.

ely.miranda@ifpi.edu.br

15

15

Sobre a “transpilação”

- O arquivo .js gerado não é sobrescrito caso existam erros de compilação:
 - O código exibe resultados anteriores, quando na verdade não houve êxito na transpilação.
 - “Trava de compilação” mediante erros.

ely.miranda@ifpi.edu.br

16

16

Sintaxe

- TypeScript é case sentive;
- Apesar de não obrigatório o uso, a comunidade JavaScript ainda utiliza ponto-e-vírgula;
- A saída padrão que usaremos será a console.log:

```
console.log("Hello, " + name);
```

ely.miranda@ifpi.edu.br

17

17

Sintaxe

- Comentários de uma linha:

```
// comentário de uma linha
```
- Comentário de várias linhas:

```
/* este é um comentário  
de mais de uma linha */
```

ely.miranda@ifpi.edu.br

18

18

Alguns tipos suportados

- number;
- string;
- boolean;
- Object;
- Date;
- Array;
- Map;
- function;
- Set;
- any;
- void;
- null;
- never.

ely.miranda@ifpi.edu.br

19

19

Além dos tipos anteriores

- Tuple;
- Enum;
- Interfaces;
- Union types;
- Type aliases.

<https://www.typescriptlang.org/docs/handbook/2/everyday-types.html>

ely.miranda@ifpi.edu.br

20

20

Declaração de variáveis

- Declarando uma variável:

```
let a : number = 10;  
console.log(a); // 10
```

- Declarando uma variável sem inicialização:

```
let a;  
console.log(a); // undefined
```

ely.miranda@ifpi.edu.br

21

21

Declaração de variáveis

- Declarando uma variável com tipo implícito (forma padrão do JS):

```
let a = 10;  
console.log(typeof a) // number
```

ely.miranda@ifpi.edu.br

22

22

Type inference x annotation

- A declaração de tipo explícita também é conhecida como **type annotation**;
- Quando não definimos explicitamente o TypeScript usa a **type inference**;
- Minha sugestão: prefira type annotation ou definição explícita dos tipos.

ely.miranda@ifpi.edu.br

23

23

Tipos básicos

- Possui os 3 tipos básicos do JS: string, number e boolean;
- O operador typeof ou função typeof() retorna o tipo de uma variável:

```
const dobra = (x: number) => 2 * x;  
console.log(typeof(dobra)); //function
```

ely.miranda@ifpi.edu.br

24

24

Tipos básicos

- **string**: cadeia de caracteres;
- **number**: engloba tipos numéricos inteiros e de ponto flutuante;
- **boolean**: tipo lógico com valores true ou false.

ely.miranda@ifpi.edu.br

25

25

Tipos básicos

- Os nomes de tipo String, Number e Boolean (começando com letras maiúsculas) são válidos;
- Tratam-se de tipos internos especiais;
- Segundo a documentação, devemos sempre usar **string**, **number** ou **boolean** para tipos.

ely.miranda@ifpi.edu.br

26

26

Tipos básicos

- boolean:
`let isDone: boolean = false;`
`isDone = true;`

ely.miranda@ifpi.edu.br

27

27

Tipos básicos

- number: engloba os tipos inteiros e de ponto flutuante:
`let inteiro: number = 6;`
`let real: number = 4.8;`
`let hex: number = 0xf0a0d;`
`let binario: number = 0b101010;`
`let octal: number = 0o344;`

ely.miranda@ifpi.edu.br

28

28

Tipagem forte

- Quando um tipo é definido, não é possível atribuir o valor de outro tipo e ocorre um erro de compilação:

```
let x: number = 10;  
x = "oi"; //o tipo 'string' não pode ser atribuído ao tipo  
'number'
```

29

29

Sobre tipagem

- Para quem tem dúvidas sobre:
 - Tipagem estática x dinâmica;
 - Tipagem fraca x forte;
- Segue um link para breve explicação:
 - <https://www.treinaweb.com.br/blog/quais-as-diferencas-entre-tipagens-estatica-ou-dinamica-e-forte-ou-fraca>

ely.miranda@ifpi.edu.br

30

30

Métodos em tipos básicos

- Os tipos básicos citados possuem alguns métodos associados:

```
let y : number = 2.590923;  
console.log(y.toPrecision(4)); //2.591
```

```
let s : string = 'ELY';  
console.log(s.toLowerCase()); //ely
```

ely.miranda@ifpi.edu.br

31

31

Tipo string

- Armazena cadeia de caracteres;
- A princípio, podem ser delimitadas aspas simples e duplas;
- Não há uma formal discriminação entre aspas simples ou duplas:

```
let color: string = "azul";  
color = 'vermelho' + " escuro";
```

ely.miranda@ifpi.edu.br

32

32

Tipo string

- Há também as template strings ou strings interpoláveis delimitadas por crases:

```
let s : string = 'ELY';  
let greeting = `Hello, ${s}`;  
console.log(greeting); // Hello, ELY
```

ely.miranda@ifpi.edu.br

33

33

Tipo string

- Template strings podem ter múltiplas linhas sem necessidade de concatenação:

```
let nome : string = "Ely Miranda";  
let idade: number = 39;  
let frase: string = `Meu nome é ${nome}.  
Completarei ${ idade + 1 } mês que vem.`;  
console.log(frase);
```

ely.miranda@ifpi.edu.br

34

34

Miscelânea: Arquivo tsconfig.json

- É o arquivo de configuração de como o TypeScript é compilado;
- Para criar o arquivo tsconfig.json “padrão”, digitamos no prompt:

`tsc --init` →

Created a new tsconfig.json with:

```
target: es2016
module: commonjs
strict: true
esModuleInterop: true
skipLibCheck: true
forceConsistentCasingInFileNames: true
```

ely.miranda@ifpi.edu.br

35

35

Miscelânea: compilação automática

- A ideia é a cada vez que salvarmos um arquivo .ts um observador (watcher) faça a transpilação automática;
- Para isso, utilizamos o comando em um prompt “a parte”:
`tsc -w`
- Já o comando do node pode ser executado normalmente em outro prompt de comando.

ely.miranda@ifpi.edu.br

36

36

Configurando a compilação automática

- Nota:
 - Caso existam erros de compilação, a última versão correta transpilada será mantida;
 - Erro muito comum nos inícios dos estudos.

ely.miranda@ifpi.edu.br

37

37



Typescript

Introdução, declaração de variáveis e tipos básicos de dados

Ely – ely.miranda@ifpi.edu.br

38