

TypeScript

Programação para Internet I
IFB - Tecnologia em Sistemas para Internet

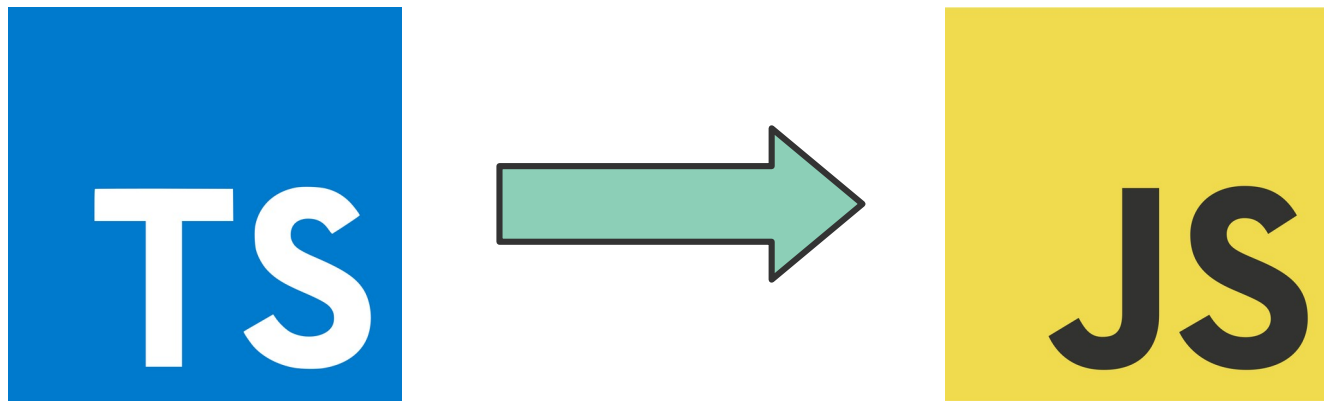
Marx Gomes van der Linden

TypeScript

- Linguagem de programação **fortemente tipada** desenvolvida pela Microsoft
 - » Software livre
- Código escrito em JavaScript tende a ser também válido em TypeScript, com pouca ou nenhuma modificação

TypeScript

- Código TypeScript **transpila** para JavaScript



- Projetado para facilitar a manutenção de aplicações de grande porte
 - » Facilidades de programação com o VS Code
 - » Permite encontrar bugs durante o desenvolvimento

TypeScript

- Para instalar:

```
npm install -g typescript
```

- Para converter um arquivo TypeScript em JavaScript:

```
tsc arquivo.ts
```

ou

```
npx tsc arquivo.ts
```

- › Automaticamente gera o **arquivo.js**

Exemplo

```
function saudacao(nome) {  
    return `Olá, ${nome}!`;  
}  
  
let meunome = "Fulano de Tal";  
  
console.log( saudacao(meunome) );
```

- Por padrão, TypeScript gera código EcmaScript 3 (máxima compatibilidade)
- Para gerar código EcmaScript 2015:

```
tsc arquivo.ts -t ES2015
```

ts-node

- Para automatizar a compilação e execução, pode-se usar o ts-node

```
npm install -g ts-node
```

- Depois, basta executar o **ts-node** no lugar do **node**:

```
ts-node arquivo.ts
```

tsconfig.json

- Todas as opções e configurações TypeScript de um projeto podem ser armazenadas no arquivo **tsconfig.json**
- Para criar o arquivo:

```
tsc --init
```

- Uma vez que o **tsconfig.json** existe no diretório, para compilar todo o projeto:

```
tsc
```

Tipos

- Variáveis em TypeScript podem ter tipos
- O compilador não permite que tipos incompatíveis sejam usados

Tipo da variável

Tipo de retorno da função

```
function saudacao(nome : string) : string{  
    return `Olá, ${nome}!`;  
}
```

```
let meunome = "Fulano de Tal";
```

```
console.log( saudacao(meunome) );
```


Tipos

- Os tipos básicos em TypeScript são:
 - » **number**
 - » **string**
 - » **boolean**
 - » **enum**
 - » **void**
 - » **any** ← **Padrão**

Tipos explícitos e implícitos

```
function saudacao(nome : string, idade : number) {  
    console.log(  
        `Olá, ${nome}! Você tem ${idade} anos.`  
    );  
}
```

```
let nome1 : string = 'Fulano de Tal';  
let nome2 = 'Beltrano da Silva';
```

```
let idade1 : number = 21;
```

```
let idade2;  
idade2 = 25;
```

→ Tipo implícito
any

```
saudacao(nome1, idade1);  
saudacao(nome2, idade2);
```

Arrays com tipos

- Um array definido com tipo só pode conter elementos daquele tipo.

```
let valores: number[] = [1, 4, 7];  
valores[1] = 5;  
valores[2] = 'abc'; ← Erro!
```

- É possível definir arrays com mais de um tipo:

```
let valores: (number | string)[] = [1, 4, 7];  
valores[1] = 5;  
valores[2] = 'abc';
```

Tuplas

- Uma tupla é um tipo de dados que define um array com número e tipos fixos de elementos

```
let valores: [string, number];  
valores = ['abc', 123];  
valores = ['abc', '123']; ← Erro!
```

Enum

- A palavra-chave enum define um tipo de dados com uma lista de valores possíveis.

```
enum Situacao {  
    Matriculado, Concluido, Evadido, Jubilado  
}
```

```
let sit1 : Situacao = Situacao.Matriculado;  
let sit2 = Situacao.Concluido;
```

Interfaces

- A interface de um objeto define que propriedades, no mínimo, ele tem
- Pode ser especificada no tipo da variável:

```
function escreveUsuario(u: { nome: string, idade: number }) {  
    console.log(`O usuário ${u.nome} tem ${u.idade}`);  
}
```

```
let usuario = {  
    nome: 'Fulano',  
    idade: 28,  
    email: 'fulano@lalalal.com'  
};
```

```
escreveUsuario(usuario);
```

Interfaces

- Pode ser declarada separadamente:

```
interface Usuario {  
  nome: string,  
  idade: number  
}  
  
function escreveUsuario(u: Usuario) {  
  console.log(`O usuário ${u.nome} tem ${u.idade}`);  
}  
  
let usuario = {  
  nome: 'Fulano',  
  idade: 28,  
  email: 'fulano@lalalal.com'  
};  
  
escreveUsuario(usuario);
```

Propriedades opcionais

- Uma interface pode ter propriedades opcionais, declaradas com o caractere ?

```
interface Usuario {  
    nome: string,  
    idade: number,  
    email?: string  
}
```


Verificação de tipos

- A verificação de tipos em TypeScript é feita pelo **conteúdo** de uma interface, não pelo seu nome
 - » Tipagem estrutural ou *duck typing*
- Interfaces anônimas ou com nomes diferentes podem ser compatíveis

```
interface Casa {  
  metragem: number,  
  numeroQuartos: number  
}
```

```
interface Apartamento {  
  metragem: number,  
  numeroQuartos: number,  
  andar: number  
}
```

```
function escreveCasa(m : Casa){  
  console.log(`${m.numeroQuartos} quartos, `  
    +`${m.metragem} metros quadrados`);  
}
```

```
let minhaCasa : Casa = { metragem: 120, numeroQuartos: 3 };  
escreveCasa(minhaCasa);
```

```
let meuApartamento : Apartamento = {  
  metragem: 90, numeroQuartos: 2, andar: 15  
}  
escreveCasa(meuApartamento);
```

Repetindo o mesmo nome

- É possível declarar múltiplas interfaces com o mesmo nome, desde que elas não sejam incompatíveis entre si.
 - » As definições são mescladas em uma só
- Pode ser útil para estender a compatibilidade de interfaces definidas em arquivos externos

```
interface Pessoa {  
  nome: string,  
  idade: number  
}
```

```
interface Pessoa {  
  nome: string,  
  sobrenome: string  
}
```

```
let p: Pessoa = {  
  nome: 'Fulano',  
  sobrenome: 'de Tal',  
  idade: 25  
}
```

```
console.log(p);
```

Tipos de retorno de funções

- Além de tipos para os parâmetros, funções também podem ter tipos de retorno:

```
function multiplica(a: number, b: number) : number {  
    return a * b;  
}
```

```
function maiorQueZero(x: number) : boolean {  
    return x > 0;  
}
```

```
console.log( multiplica(10,4) );  
console.log( maiorQueZero(5) );  
console.log( maiorQueZero(-8) );
```

Em funções anônimas

- Funções anônimas também podem ter tipos de retorno:

```
let itens: number[] = [4, 10, 8];
```

```
let dobra: (n: number) => number;  
dobra = n => n*2;
```

```
let metade: (n: number) => number = n => n/2;
```

```
let outra: (s: string) => number;  
outra = s => parseFloat(s) + 10;
```

```
console.log( itens.map(dobra) );  
console.log( itens.map(metade) );  
console.log( itens.map(outra));
```

← Erro!

Parâmetros opcionais

- Parâmetros opcionais usam a mesma sintaxe das interfaces

```
function saudacao(nome: string, idade?: number) : string{  
  if (idade)  
    return `Olá, ${nome}! Você tem ${idade} anos.`;  
  else  
    return `Olá, ${nome}`;  
}
```


```
console.log( saudacao ('Fulano') );  
console.log( saudacao ('Beltrano', 25) );
```

Parâmetros com opções de tipo

- É possível definir parâmetros em que mais de um tipo é possível:

```
function escreve(codigo: (number | string)): void {  
  console.log(`Código: ${codigo}`);  
}
```

```
console.log(123);  
console.log('456');
```



Significa que a função
não retorna nada