

TypeScript

Introdução, declaração de variáveis, tipos de dados e funções

Ely – elydasilvamiranda@gmail.com

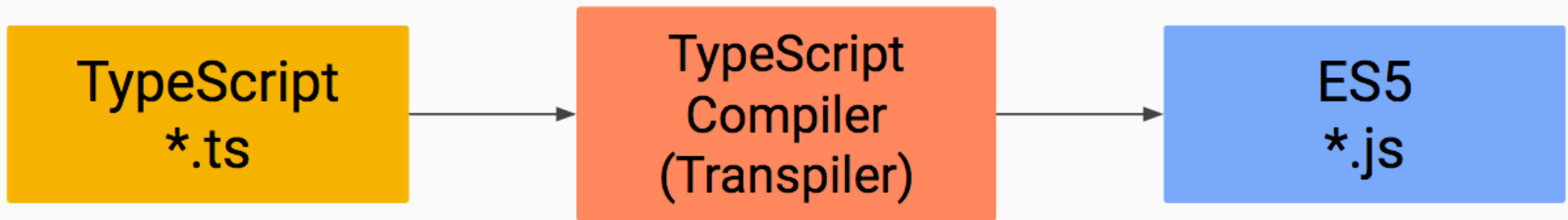
TypeScript

- Linguagem de programação open source desenvolvida pela Microsoft;
 - Mesmo criador do C# e Delphi;
- É um super conjunto de JavaScript;
- Adiciona ao JS elementos como:
 - tipagem estática explícita;
 - Classes, generics e interfaces;
 - anotações...;
- Site oficial: <https://www.typescriptlang.org/>



No navegador

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



<http://anthonygiretti.com/2017/06/04/no-typescript-is-not-compiled-into-javascript>



<https://blog.sessionstack.com/how-javascript-works-the-internals-of-classes-and-inheritance-transpiling-in-babel-and-113612cdc220>

**Por que (o ecossistema)
JavaScript?**

Por que JavaScript?

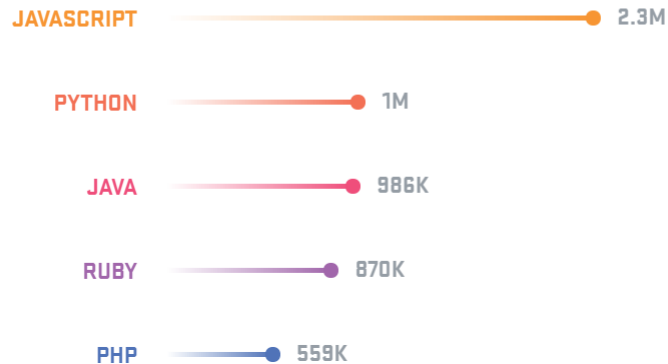
- JavaScript teve sua aplicação ampliada, não só no front-end de aplicações.



The fifteen most popular languages on GitHub

by opened pull request

GitHub is home to open source projects written in 337 unique programming languages—but especially JavaScript.



<https://octoverse.github.com/2017/>

GitHub

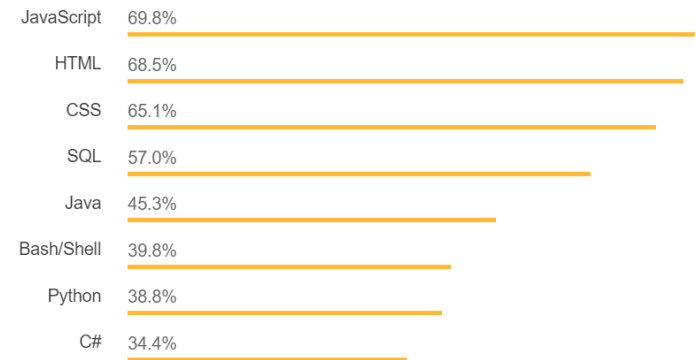


Most Popular Technologies

Programming, Scripting, and Markup Languages

All Respondents

Professional Developers



<https://insights.stackoverflow.com/survey/2018>

Por que não apenas JavaScript?

Alguns desafios do JavaScript

- Orientação a objetos “estranha” e limitada, com uso de prototypes e sem uma sintaxe adequada;

```
function Pessoa(nome, sobrenome, idade) {  
    this.nome = nome;  
    this.sobrenome = sobrenome;  
    this.idade = idade;  
}
```

```
var p1 = new Pessoa("Joao", "Silva", 40);  
var p2 = new Pessoa("Aline", "Sousa", 38);
```

- ... Pesquisem como se faz uma Herança...

Alguns desafios do JavaScript

- Tipagem estática leva a erros de runtime;

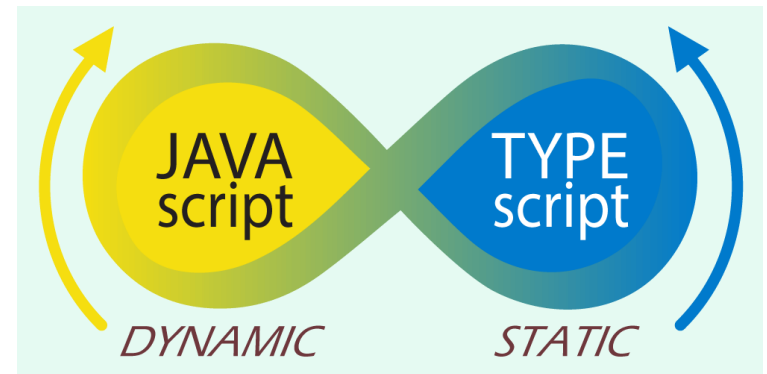
```
function soma(x, y) {  
    return x + y;  
}
```

```
alert(soma(1, 2)); // 3
```

```
alert(soma(1, "2")); // "12"
```


Alguns desafios do JavaScript

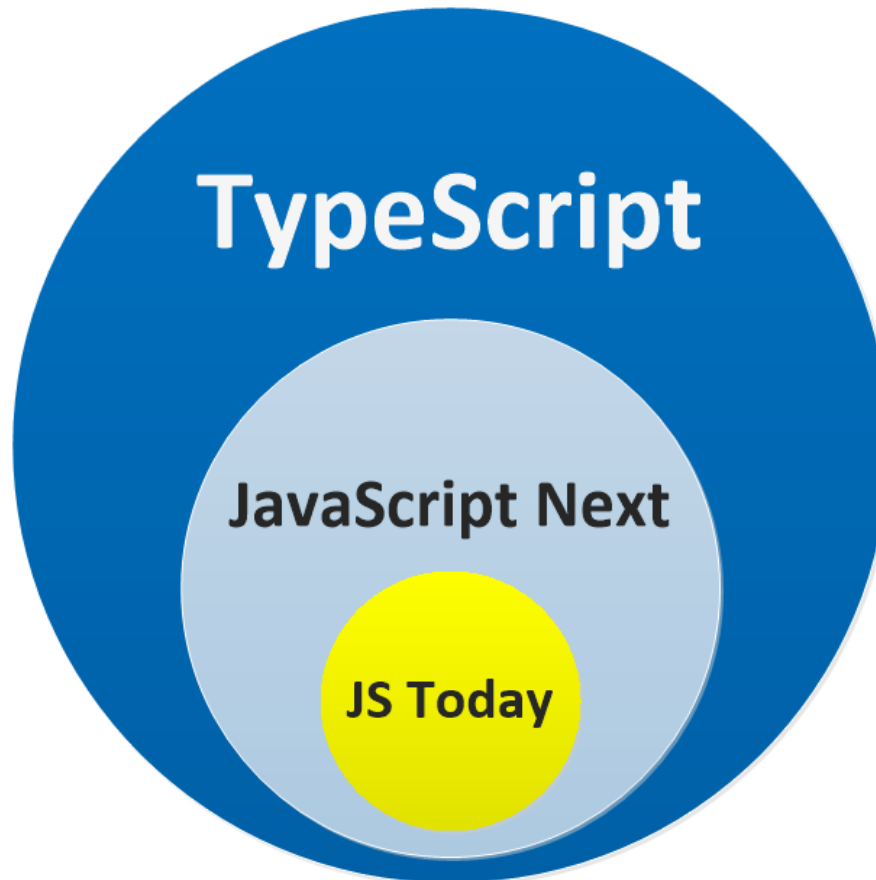
- Com tipagem dinâmica e implícita:
 - Pouco se pode inferir sobre as construções em tempo de projeto;
 - Pouco se pode avançar em ferramentas como:
 - “Linters”;
 - Typecheckers;
 - Intellisenses.
- No TypeScript a verificação de tipos é estática.



<https://www.happilyon.com/stream.html>

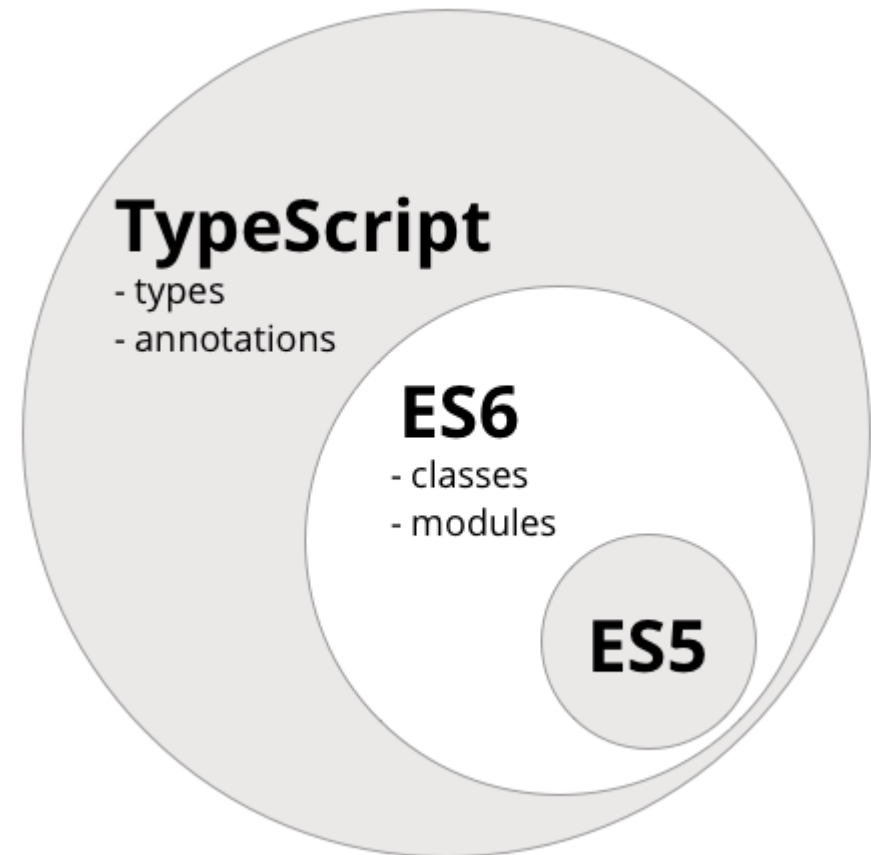
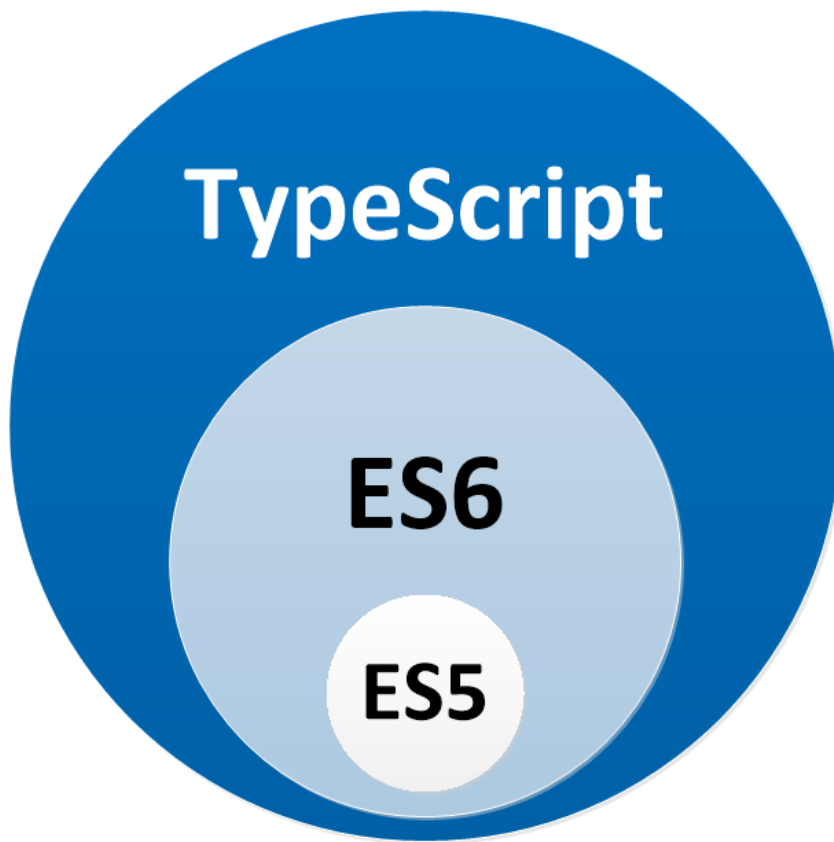
TypeScript

Um super conjunto de JavaScript
(ECMA Script 6 e 5)



TypeScript

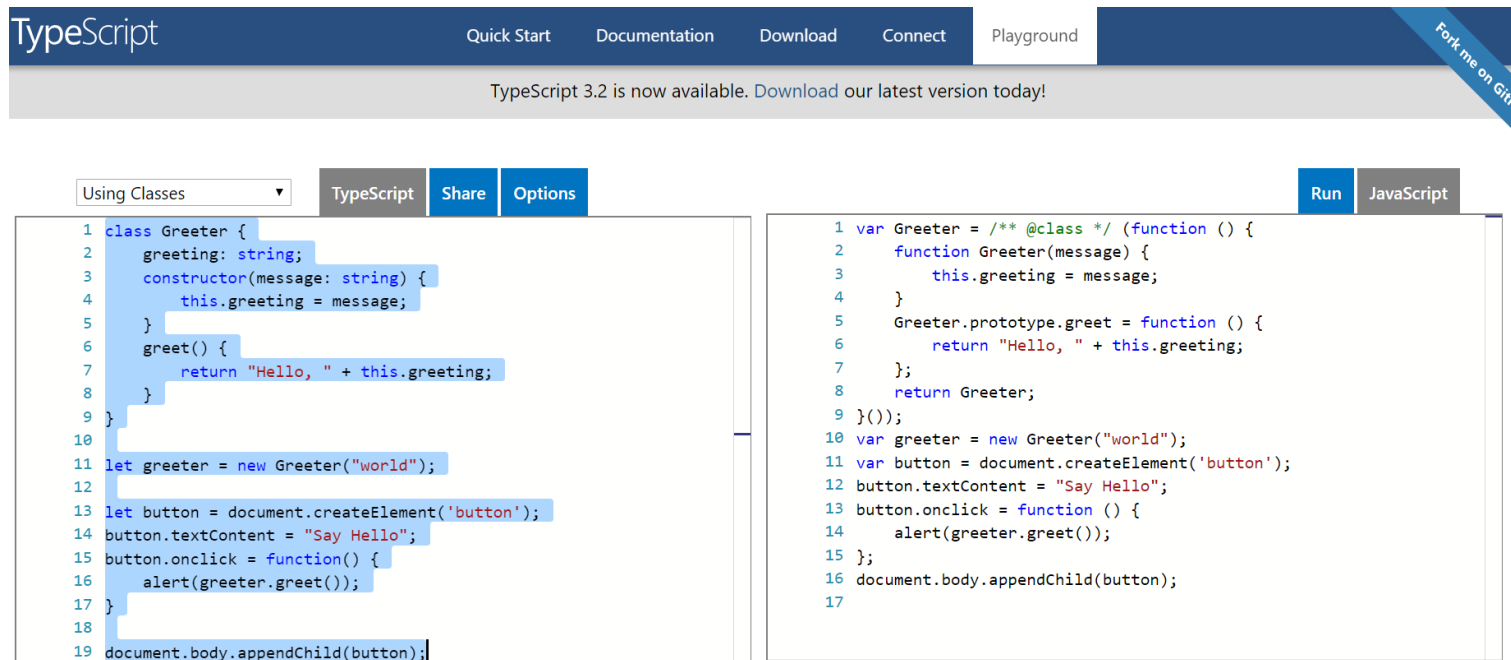
Um super conjunto de JavaScript
(ECMA Script 6 e 5)



Para curiosos

- O site abaixo traduz o código TypeScript em JavaScript em tempo real:

<http://www.typescriptlang.org/play/>



The screenshot displays the TypeScript Playground interface. At the top, there's a navigation bar with links: Quick Start, Documentation, Download, Connect, and Playground. A banner below the navigation bar states "TypeScript 3.2 is now available. Download our latest version today!". On the right side of the navigation bar, there's a "Fork me on GitHub" button. Below the navigation bar, there's a dropdown menu set to "Using Classes" and buttons for "TypeScript", "Share", and "Options". The main area is split into two panels. The left panel shows TypeScript code, and the right panel shows the corresponding JavaScript code. The "Run" and "JavaScript" buttons are visible on the right side of the code panels.

```
1 class Greeter {  
2   greeting: string;  
3   constructor(message: string) {  
4     this.greeting = message;  
5   }  
6   greet() {  
7     return "Hello, " + this.greeting;  
8   }  
9 }  
10  
11 let greeter = new Greeter("world");  
12  
13 let button = document.createElement('button');  
14 button.textContent = "Say Hello";  
15 button.onclick = function() {  
16   alert(greeter.greet());  
17 }  
18  
19 document.body.appendChild(button);
```

```
1 var Greeter = /** @class */ (function () {  
2   function Greeter(message) {  
3     this.greeting = message;  
4   }  
5   Greeter.prototype.greet = function () {  
6     return "Hello, " + this.greeting;  
7   };  
8   return Greeter;  
9 }());  
10 var greeter = new Greeter("world");  
11 var button = document.createElement('button');  
12 button.textContent = "Say Hello";  
13 button.onclick = function () {  
14   alert(greeter.greet());  
15 };  
16 document.body.appendChild(button);  
17
```

Motivação principal

- O principal objetivo do TypeScript é prover um melhor sistema de tipos;
- Consequências:
 - Checagem de tipos;
 - Desenvolvimento de ferramentas como intellisense, typechecks etc;
 - Warnings e erros evitando que bugs sejam descobertos apenas em runtime; (...pelos usuários);
- Resumindo, busca-se **mais** de produtividade.

Motivação principal

Why TypeScript?



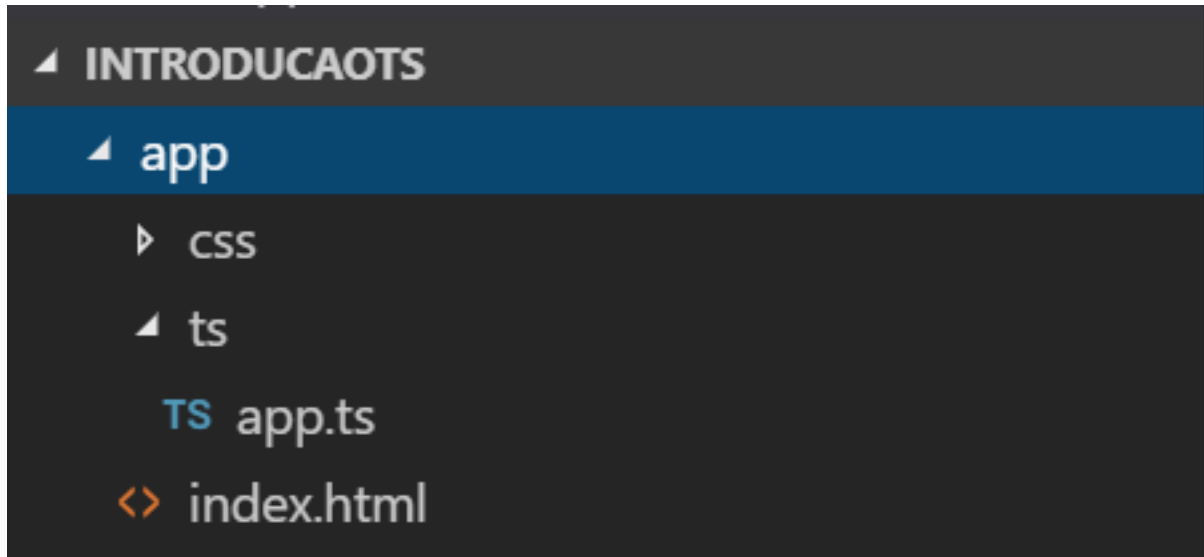
*We love TypeScript for many things... With TypeScript, several of our team members have said things like '**I now actually understand most of our own code!**' because they can easily traverse it and understand relationships much better. And we've found several bugs via TypeScript's checks."*

— Brad Green, Engineering Director - AngularJS

<https://www.typescriptlang.org/>

Criando um projeto

- Descompacte o arquivo introducaots.rar
- Estrutura inicial:



Criando um projeto

- Usaremos a infraestrutura do Node para gerenciar nossos scripts;
- Dentro da pasta **introducaots**, execute o comando:

 >> npm init
- Confirme todas as “perguntas” com <enter>;
- Com isso, é criado um arquivo package.json que será usado para a configuração do projeto.

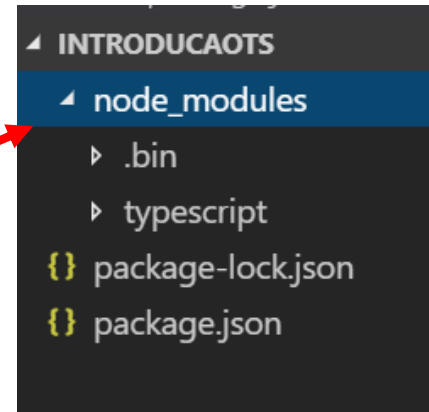
Arquivo package.json

```
{  
  "name": "introducaots",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit  
1",  
  },  
  "author": "",  
  "license": "ISC",  
}
```

Adicionando o TypeScript ao projeto

- Ainda no prompt, execute o comando:
>> npm install typescript@2.3.2 --save-dev

- Uma versão “lite” do typescript é baixada e configurada em node_modules:




- E a seguinte configuração é adicionada automaticamente ao package.json:

```
-----  
  "devDependencies": {  
    "typescript": "^2.3.2"  
  }
```

Ainda no package.json

- Adicione o compilador do typescript ao arquivo package.json:tsconfig.json:

```
{
  "name": "introducaots",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "compile": "tsc"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "typescript": "^2.3.2"
  }
}
```



Entendendo o arquivo tsconfig.json

- Arquivos *.ts de **app/ts** e **subpastas** serão transpilados para JavaScript **ES 5**
- Arquivos *.js serão salvos no mesmo diretório;
- Caso haja erro de compilação, os arquivos JS não serão gerados.

```
{
  "compilerOptions": {
    "target": "es5",
    "noEmitOnError": true,
    "sourceMap": true
  },
  "include": [
    "app/ts/**/*"
  ]
}
```

O arquivo app.ts

```
let nome : string = "Type Script";  
console.log("Hello, " + nome);
```


Compilando o projeto

- Na pasta introducaots execute o comando:
>> npm run compile
- Na verdade, esse comando chama o compilador “tsc” conforme configurado no package.json;
- O resultado é a criação do arquivo **app.js** e **app.js.map** no diretório;
- Arquivo app.js.map:
 - Serve é um mapeamento das instruções do app.ts para as instruções do arquivo app.js.

Automatizando a compilação

- Basta adicionar a linha no arquivo package.json

```
{
  "name": "introducaots",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "compile": "tsc",
    "start": "tsc -w"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "typescript": "^2.3.2"
  }
}
```



- Rodar o comando:

>> npm start

Sintaxe

- TypeScript é case sensitive;
- Apesar de não obrigatório o uso, a comunidade JavaScript ainda utiliza ponto-e-vírgula;

- Comentários de uma linha:

`// comentário de uma linha`

- Comentário de várias linhas:

`/* este é um comentário`

`de mais de uma linha */`

- A saída de dados padrão que usaremos será:
`console.log()`.

Alguns tipos suportados

- number;
- string;
- boolean;
- Object;
- Date;
- Array;
- Map;
- Set;
- Tuple;
- Enum;
- any;
- void;
- null;
- never;

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

Declaração de variáveis

- Declarando uma variável:

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

- Declarando uma variável sem inicialização:

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

- Inspecionando o tipo de uma variável com tipo implícito (forma padrão do JS):

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

Declaração de variáveis

- Declarando uma variável e tentando atribuir um tipo diferente:

```
let a : number = 10;
```

```
a = "ely"; //não compila
```

```
error TS2322: Type '"ely"' is not assignable to type  
'number'.
```

Declaração de variáveis

- O tipo any deixa a checagem de tipos semelhante ao JavaScript:

```
let a : any = 10;  
a = "ely";  
console.log(a); //saída: ely
```

- Entretanto, a predominância de uma tipagem forte é uma premissa do TypeScript;
- Dessa forma, o uso do tipo any deve ser usado apenas em casos muito bem justificados.

Sobre tipagem

- Para quem tem dúvidas sobre:
 - Tipagem estática x dinâmica;
 - Tipagem fraca x forte;
- Segue um link para breve explicação:
 - <https://goo.gl/bDSXnV>

Tipos básicos

- boolean:

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

```
isDone = true;
```

- number: engloba os tipos inteiros e reais

```
let inteiro: number = 6;
```

```
let real: number = 4.8;
```

```
let hex: number = 0xf0a0d;
```

```
let binario: number = 0b101010;
```

```
let octal: number = 0o344;
```

Tipos básicos

- string: cadeias de caracteres delimitadas por “” e “”

```
let color: string = "azul";
```

```
color = 'vermelho' + " escuro";
```

- Não há uma distinção explícita sobre o uso de aspas simples ou duplas;
- Há ainda um terceiro tipo de “marcação” para strings: as template strings.

Template Strings

- Podem ter múltiplas linhas sem necessidade de caracteres de escape;
- Também podem conter expressões;
- São delimitadas por **crases** ``:

```
let nome : string = `Ely Miranda`;
```

```
let idade: number = 39;
```

```
let frase: string = `Meu nome é ${nome}.
```

```
Complearei ${ idade + 1 } mês que vem.`;
```

```
console.log(frase);
```


Enums

- Como Java ou C#, são maneiras de fornecer nomes para conjuntos de valores numéricos:

```
enum Cores {Vermelho, Verde, Azul}
```

```
let c: Cores = Cores.Verde;
```

```
console.log(c); // 1
```

```
console.log(Cores[2]) //azul
```

Arrays

- Podem ser declarados apenas seguidos de []:
`let numeros: number[] = [1, 2, 3];`
- Podem ser declarados como “arrays tipados”:
`let numeros: Array<number> = [1, 2, 3];`
- Arrays são objetos e possuem alguns métodos:
`let numeros: number[] = [1, 2, 3];`
`numeros.push(4);`
`console.log(numeros.reverse()); // [4,3,2,1]`

Ainda sobre tipos

- E os tipos Number, String...
- Em TypeScript são objetos:

```
let cidade = new String('Teresina');  
console.log(typeof(cidade)); // Object  
let ano = new Number(50);  
console.log(typeof(ano)); // Object
```

Ainda sobre tipos

- Não é comum usarmos os objetos Number, String etc;
- É feito um auto-boxing caso necessário:

```
let cidade : string = 'Teresina';  
console.log(typeof(cidade)); // string  
let ano : number = 50.3333;  
console.log(typeof(ano)); // number  
  
console.log(ano.toFixed(2)); //50.33  
console.log(cidade.concat(" - THE")); //Teresina - THE
```

let ou var

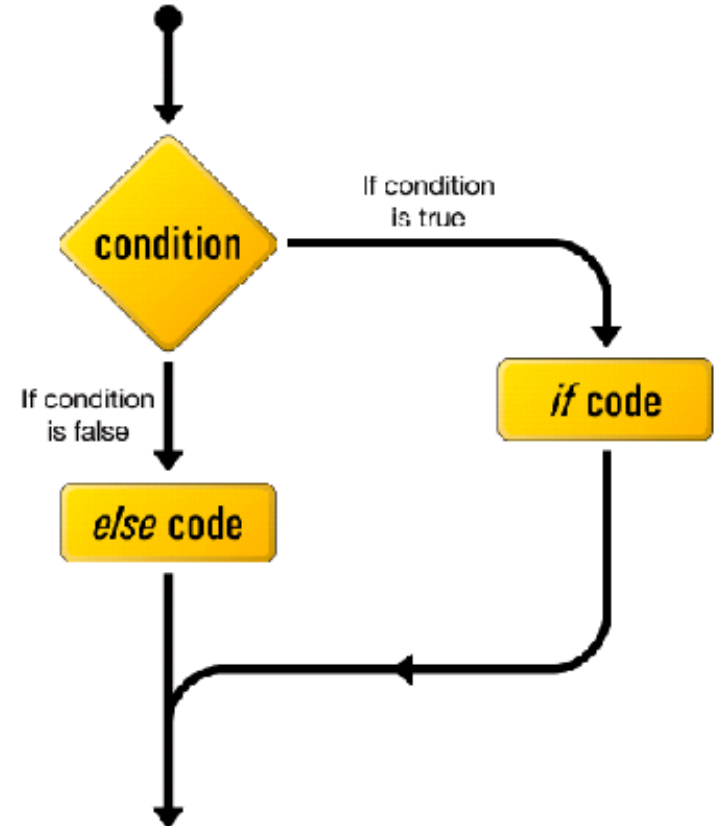
- Também é possível declarar variáveis com as palavras reservadas var e const;
- Usando const, a variável não pode ter seu valor alterado;
- Usando var, o escopo da variável é mais amplo, já com let o escopo é mais restrito:

```
for (let i : number = 0; i < 3; i++)  
    console.log(i); // 1, 2, 3  
console.log(i); // undefined  
for (var i : number = 0; i < 3; i++) {  
    console.log(i); // 1, 2, 3  
console.log(i); // 3
```

Estruturas de decisão – if e else

- Sintaxe:

```
if (condição) {  
    código da condição verdadeira;  
}  
else {  
    código da condição falsa;  
}
```



{ simboliza um início/begin

}

} representa um fim/end

Operadores condicionais e lógicos

>	A > B
>=	A >= B
<	A < B
<=	A <= B
==	A == B
!=	A != B

← A é igual a B

← A é diferente de B

- && : and
- || : or
- ! : not

```
let idade = 17;  
if (idade >= 16 && idade < 18) {  
  console.log("voto facultativo");  
}
```

Estruturas de decisão – Switch

```
switch (exp) {  
    case valor 1:  
        //código a ser executado se a expressão = valor 1;  
        break;  
    case valor n:  
        //código a ser executado se a expressão = valor n;  
        break;  
    default: //executado caso a expressão não seja nenhum  
dos valores;  
}
```


Expressões compactadas

- Em TypeScript também podemos utilizar formas “compactada” instruções:

```
let numero : number = 100;
```

```
numero++;
```

```
numero--;
```

```
numero += 2;
```

```
numero -= 1;
```

```
numero *= 4;
```

```
numero /= 2;
```

```
console.log(numero); // 202
```

Estruturas de repetição - while

- Executa um trecho de código enquanto uma condição for verdadeira:

```
let numero = 1;
while (numero <= 3) {
    console.log("O número atual é: " + numero);
    numero = numero + 1;
}
```

Estruturas de repetição – do...while

- Executa um trecho de código enquanto uma condição for verdadeira;
- Mesmo que a condição seja falsa, o código é executado pelo menos uma vez:

```
let numero = 1;
do {
    console.log("O número atual é: " + numero);
    numero = numero + 1;
} while (numero <= 3) ;
```

Estruturas de repetição - for

- Executa um trecho de código por uma quantidade específica de vezes:

```
let numeros = [4, 5, 6];  
  for (let i = 0; i < numeros.length; i++) {  
    numeros[i] = numeros[i]* 2;  
    console.log(numeros[i]);  
  }
```

Estrutura de repetição - for

- Além da estrutura de for e while conhecidos em Java, há dois outros tipos de for:

```
let numeros = [4, 5, 6];  
//faz a iteração pelos elementos  
for (let numero of numeros) {  
    console.log(numero); // 4, 5, 6  
}  
//faz a iteração pelos índices  
for (let numero in numeros) {  
    console.log(numero); // 0, 1, 2  
}
```

Funções

- Sintaxe:

```
function nomeDaFuncao(parametros) : tipoDeRetorno {  
    //instruções da função.  
}
```

Nota: o tipo de retorno é opcional;

- Exemplo:

```
function add(x: number, y: number): number {  
    return x + y;  
}
```

```
console.log(add(2,3));
```

Funções

- Funções como variáveis:

```
var a = function add(x: number, y: number):  
    number {  
  
    return x + y;  
  
}  
console.log(a(2,4))
```

Funções

- Parâmetros “default”:

```
function nomeCompleto(nome: string,  
                        sobrenome : string = "silva")  
    : string {  
    return nome + " " + sobrenome;  
}
```

```
console.log(nomeCompleto('ely', 'miranda'));  
// ely miranda
```

```
console.log(nomeCompleto('ely'));  
// ely silva
```


Funções

- Rest parameters: uma forma de se passar um número indeterminado de parâmetros;
- Os parâmetros são passados como um array:

```
function somar(...numeros : number[]) {  
    let soma = 0;  
    for (let numero of numeros)  
        soma += numero;  
    return soma;  
}
```

```
console.log(somar()); // 0
```

```
console.log(somar(1,2)); // 3
```

```
console.log(somar(1,2,3)); // 6
```

Arrow functions

- São funções com sintaxe simplificada;
- Dado o exemplo:

```
function dobra(x : number) : number {  
    return x*2;  
}
```

- O exemplo acima pode ser redefinido como uma arrow function:

```
var dobra = (x) => x*2;  
console.log(dobra(2)); //4
```

Arrow functions

- Outro exemplo:

```
function soma(x : number, y: number) {  
    return x+y;  
}
```

- Redefinindo com arrow function:

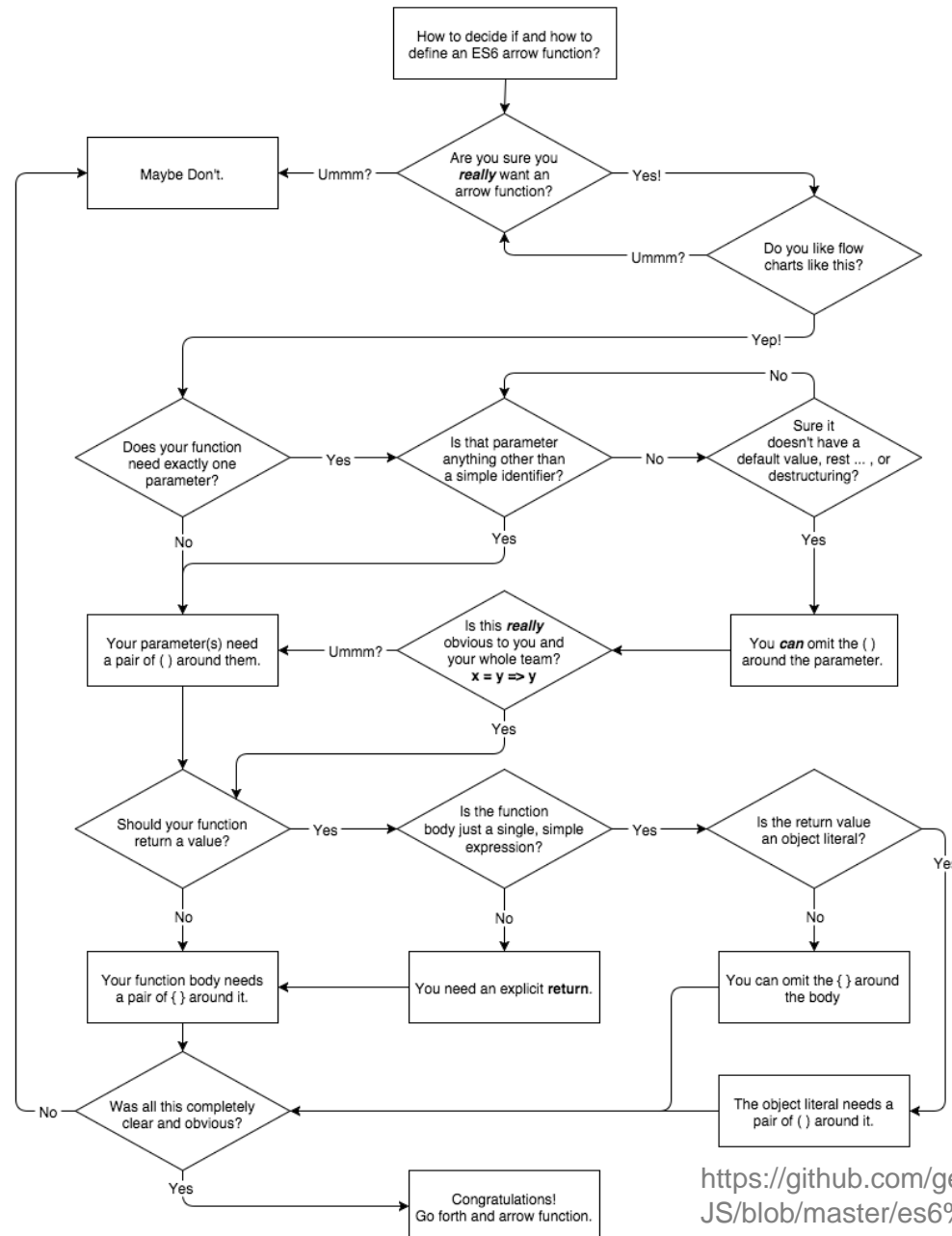
```
var soma = (x,y) => x+y;  
console.log(soma(2,4)); //6
```

Arrow functions

- É muito comum o uso com funções “iterativas” de algumas classes:

```
let numeros : number[] = [1,2,3,4];  
numeros = numeros.map(x => 2*x);  
console.log(numeros); //[2,4,6,8]
```

Quando usar uma arrow function



TypeScript

Introdução, declaração de variáveis, tipos de dados e funções

Ely – elydasilvamiranda@gmail.com