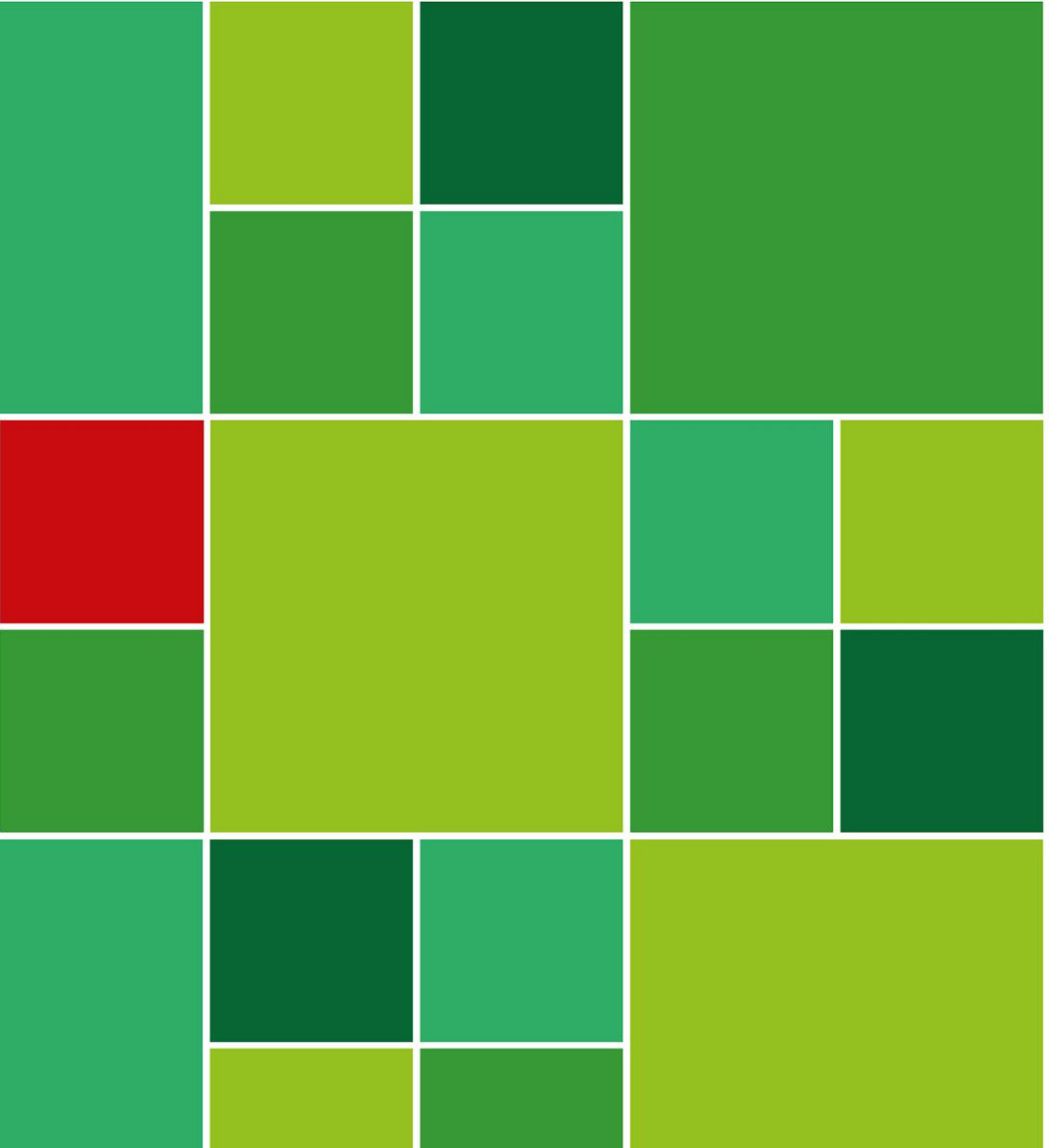


Programação WEB III

Introdução à TypeScript

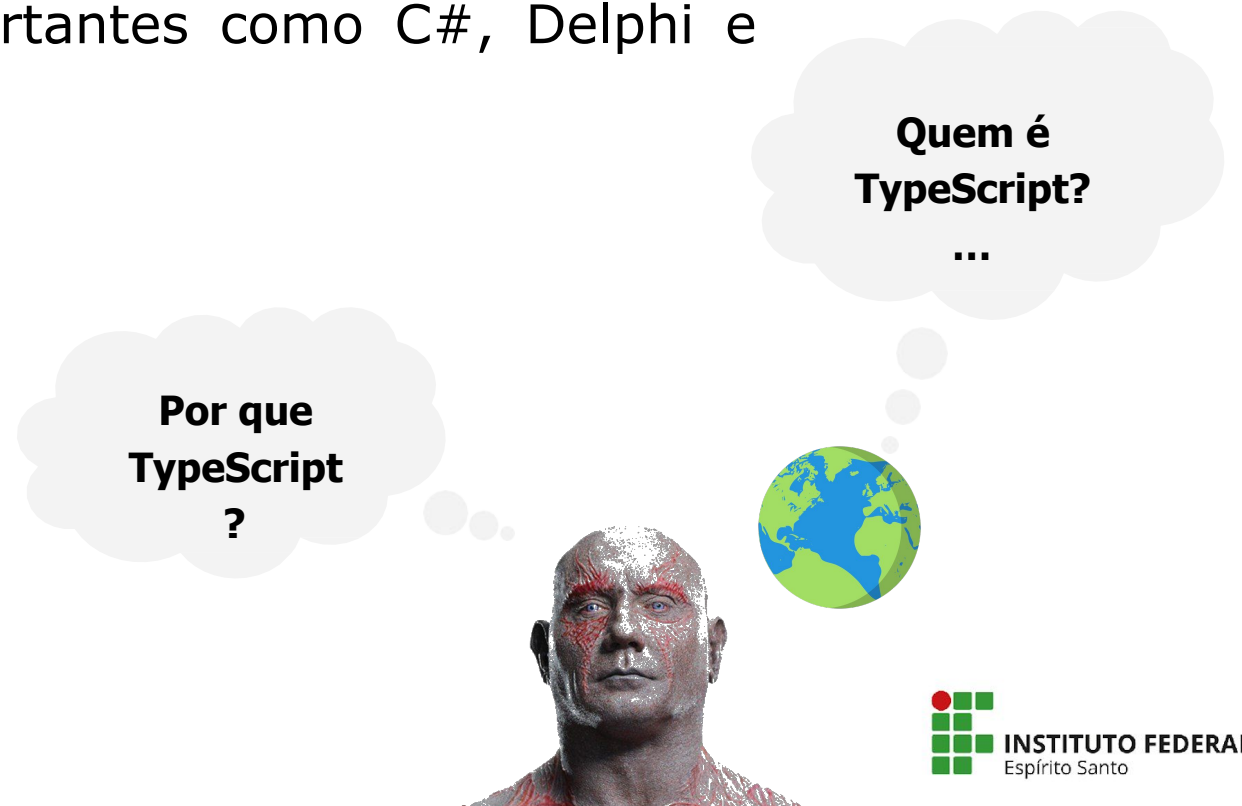
Prof. Dr. Danilo Barbosa

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PERNAMBUCO

Conteúdo

- Conceitos de TypeScript
- Tipos de TypeScript
- Referências

Lançado pela Microsoft, o TypeScript foi criado por um Arquiteto de software chamado Anders Heljsberg, que participou da criação de outras linguagens muito importantes como C#, Delphi e Pascal.



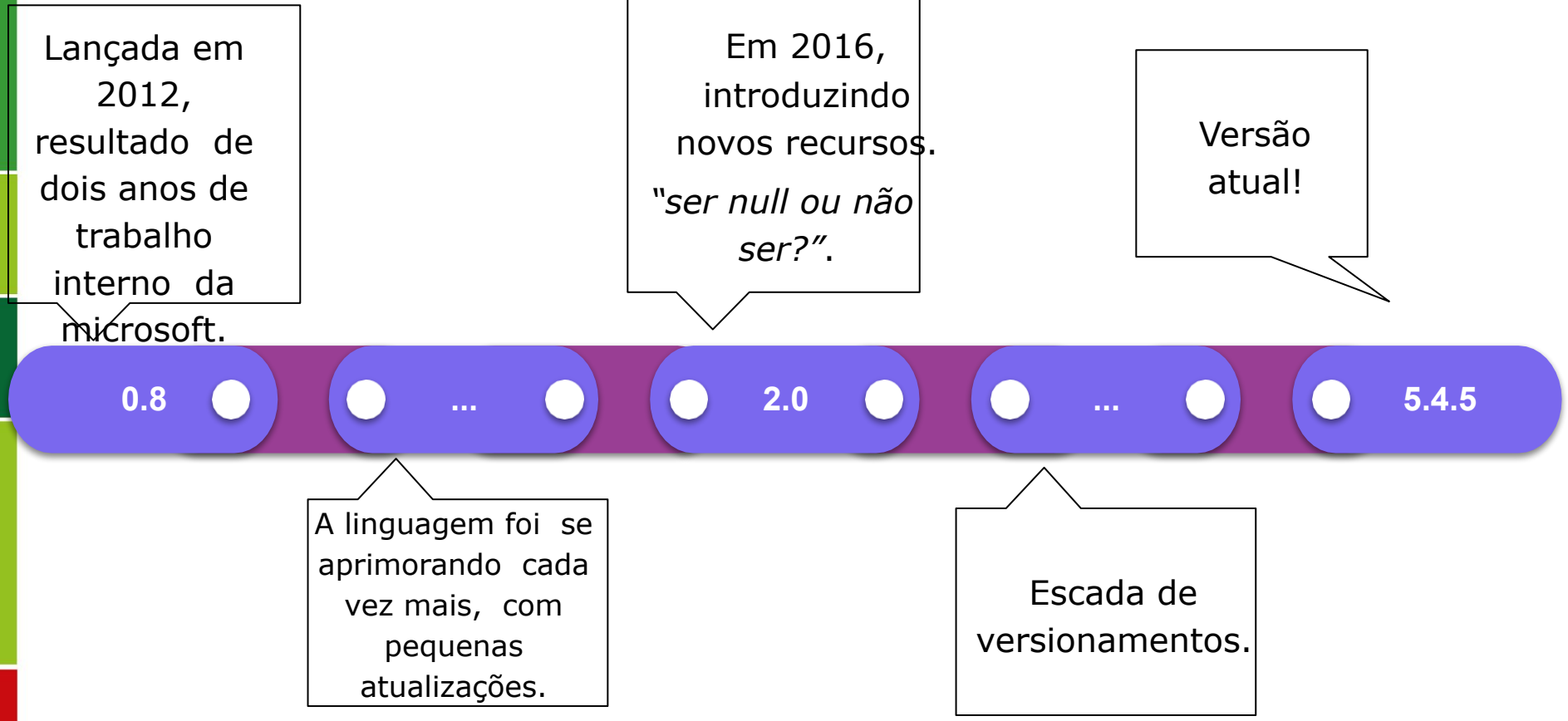
INSTITUTO FEDERAL
Espírito Santo

- **Quem é TypeScript?**

Podemos chamar TypeScript como um super conjunto da linguagem Javascript, com objetivo de elevar o nível de javascript, que antes era usado em códigos relativamente pequenos e simples.

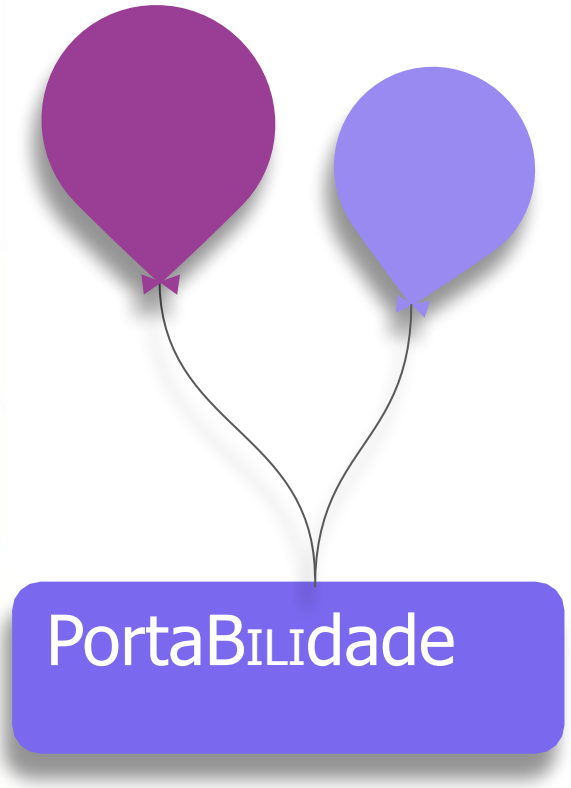
- **Por que TypeScript?**

O TypeScript trouxe consigo além da tipagem estática, o paradigma Orientação a Objeto, que facilita e viabiliza o desenvolvimento de softwares de longa “carga”, o que era requisitado na época já que o JavaScript não dava suporte.



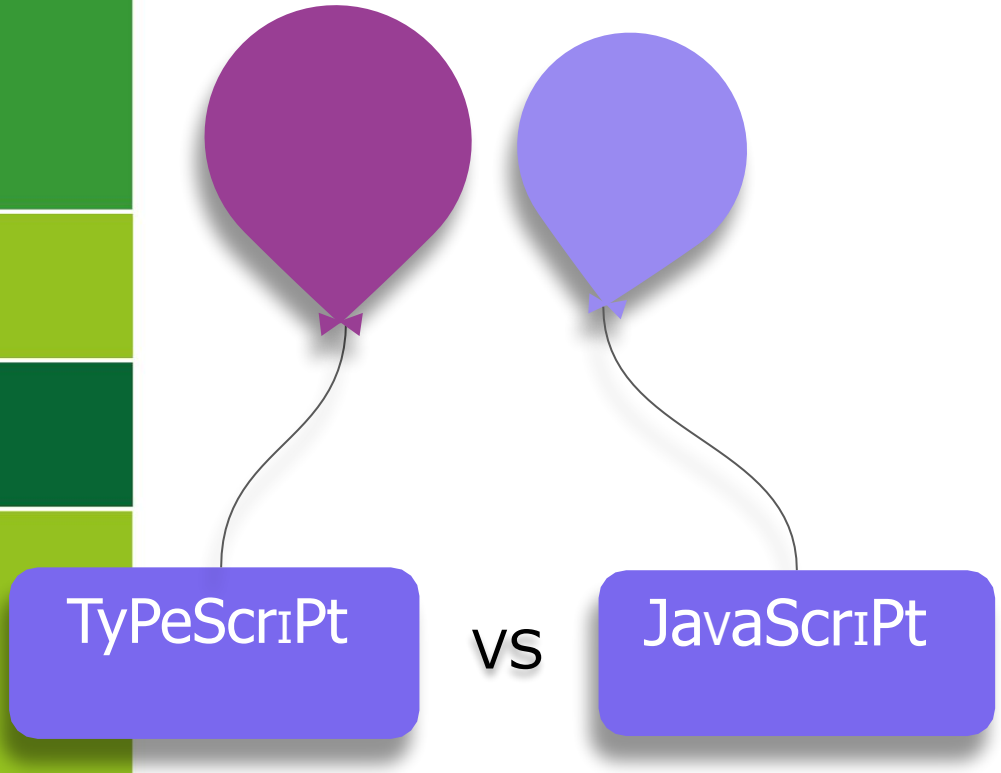
INSTITUTO FEDERAL
Espírito Santo

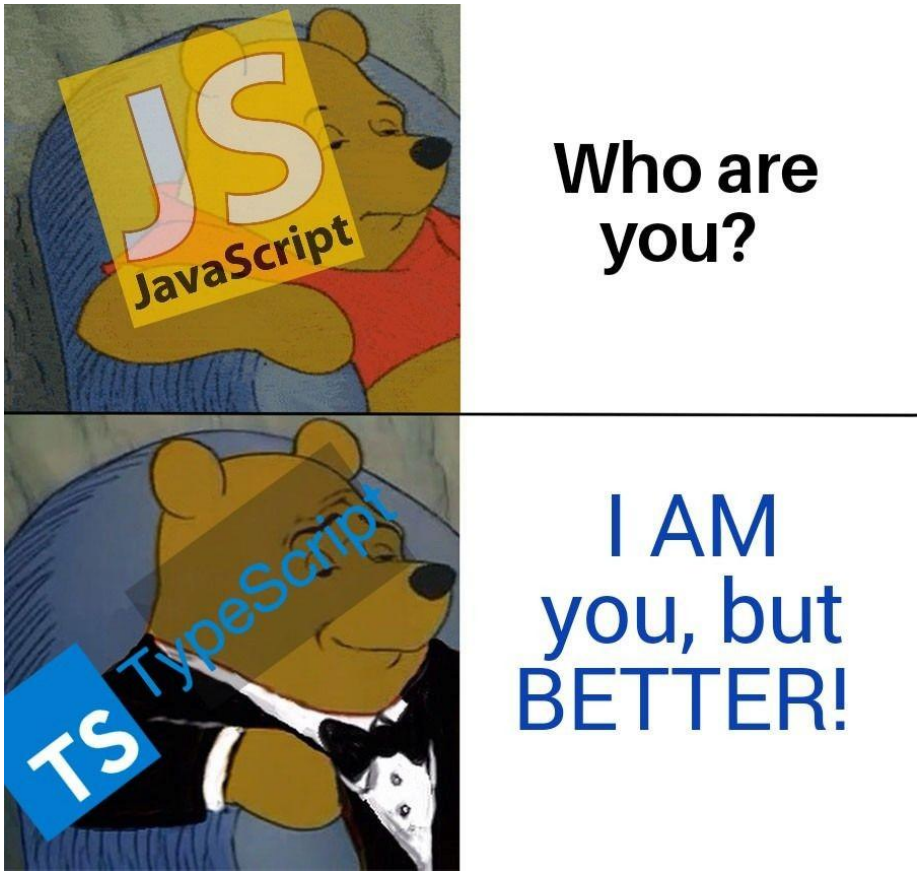
- **Por que eu deveria usarTypeScript?**
  - TypeScript está em constante aprimoramento.
  - possui uma comunidade ativa, foi introduzido e é mantido pela microsoft (as pessoas que querem colaborar tem total brecha para tal, OpenSource).
  - possui compatibilidade com JavaScript.
  - A linguagem é muito usada para desenvolvimentos web de grande porte;
  - Sua documentação é completa e sofre atualização com as novas mudanças.



- O TypeScript praticamente pode ser executado em todos navegadores em qualquer ambiente, SO e dispositivos.

- Qualquer coisa que rode JavaScript irá rodar TypeScript.
- Não é necessário uma máquina virtual pessoal ou um ambiente de tempo de execução para poder executar os códigos.


TyPeScrıPt
vs
JavaScrıPt


JS
JavaScript
Who are you?
TS
TypeScript
I AM you, but BETTER!

- TypeScript é considerada pela comunidade como uma linguagem fortemente tipada, diferente do JavaScript, TypeScript possui tanto tipagem dinâmica quanto estática.

  + Legibilidade, muito mais fácil de entender um código tipado estaticamente.

  + Confiabilidade, ao se usar tipagem estática, o compilador verifica as operação em cima dos tipos.

  - Redigibilidade, identificar sempre o tipo nas declarações e definições (o que não é muito a se perder!).

- Comunidade do JavaScript ainda é maior que a do TypeScript.
- É necessário compilar códigos TypeScript, já em JavaScript não é necessário.
- TypeScript possui a verdadeira Orientação a Objetos, enquanto JavaScript usa de recursos para simular.

- Fortemente tipada
- Orientada a Objetos e Funcional
- Suporta Tipos Genéricos
- O código é compilado para JavaScript
- TypeScript extends JavaScript
- TypeScript segue a especificação da ECMAScript 2015 (ou seja padrão ECMA-262)
- Suporte Unicode
- CaseSensitive

- **Instalando o compilador TypeScript**

A maneira mais fácil de instalar o TypeScript é por meio do npm, o Node.js Package Manager . Se você tiver o npm instalado, poderá instalar o TypeScript globalmente (-g) no seu computador por:

npm install -g typescript

tsc --version

Para compilar seu código TypeScript, você pode abrir o Terminal Integrado (Ctrl+`) e digitar tsc helloworld.ts. Isso compilará e criará um novo helloworld.jsarquivo JavaScript.

Se você tiver o Node.js instalado, poderá executar o node helloworld.js.

- **Espaços em branco e quebra de linha.**
    - TypeScript ignora os espaços em branco e quebras de linha, ou seja o programador é livre para formatar o código da maneira que achar melhor.

- **Ponto e vírgula é opcional.**

```
console.log("quero ;");
console.log("não quero ;")

console.log("aqui");
console.log("é obrigado ;");
```

- **Comentários**
  - / para comentários de uma linha
  - /* */ para comentários de várias linhas

- **Identificadores**
  - Podem conter caracteres e dígitos. No entanto, não podem começar com um dígito.
  - Não podem conter símbolos especiais, com exceção de "_" e "$", e pode conter acentuação.
  - Não podem ser palavras chaves. (break, type, private, for, finally…)
  - Não há limite de tamanho mencionado na documentação.
  - CaseSensitive.
  - Não podem conter espaços.

- **Variáveis**
  - Para declarar variáveis no TypeScript podemos usar de três formas var e const.
  - Var, let e const possuem a mesma sintaxe:

```
let <NomeDaVariável>:Tipo =Valor;
const <NomeDaVariável>:TipValor;
var <NomeDaVariável>:Tipo = Valor;
```

  - Podemos inferir ou não valores às variáveis durante sua declaraçã
  - Podemos ou não dizer o tipo de dado que ela agrega.

Exemplos :

```
const UmaConstante:  number = 2;
let UmaVariavel;

var UmaOutraVariavel = "Xananam";
let SouUmaVariávelComAcentuação = "xixiriri";
var ơ_ơ  = "????"; (SIM ISSO É VÁLIDO)
```

- Podem receber funções como valores.

```
let x = function () {
    return "Ferias!!!";
}
console.log(x());//Ferias!!!
```

```
var y = function () {
    return ":(";
}
console.log(y());//:(
```

- **Let vs Var**
  - Escopo (Var)

    Diferente do costume, declarações var possuem escopo de função.

```
function m1(): void {
    var x: boolean = true;
    if (x) {
        var loucura = "Xoxo"
    }
    console.log(loucura);//Xoxo
}
```

Ainda sobre var, existe o conceito de elevação, onde você pode declarar variáveis globais no final do código e em tempo de execução ela ser elevada ao topo. O mesmo ocorre para subprogramas.

```
function m1(): void {

console.log(opa);

}
m1();//undefined
var opa = "lelê";
m1();//lelê
```

```
function m3(): void {
console.log(x);
var x;

}
m3();//undefined
```

INSTITUTO FEDERAL
Espírito Santo

- Escopo (Let)

  Let declarações já se assemelham com a maioria das linguagens e possui escopo de bloco.

```
function m2(): void {
    let x: boolean = true;
    if (x) {
        let b;
    }
    console.log(b);//error TS2304: Cannot find name 'b'
}
```

O conceito de elevação só se aplica para let quando se trata de variáveis globais.

```
function m3(): void {
    console.log(x);
let x;
}
m3();//error escopo da
variável 'x' usada antes da
declaração
```

```
function m4(): void {
    console.log(eleve);
}
let eleve = "fui elevado";
m4();//fui elevado
```

INSTITUTO FEDERAL
Espírito Santo

- Re-declaração e sombreamento

  Utilizando var, múltiplas declarações de uma mesma variável é válida!

```
function multDeclaracao(){
    var x = 1;

    var x = 2;
    var x; // erro, x só pode ser do tipo number
    var x = "Errei denovo";//erro

}
```

INSTITUTO FEDERAL
Espírito Santo

O sombreamento em declarações var são limitadas de duas formas.

```
var q = 1;
function f() {
    var q = 2;
    console.log(q);
}
f();//2
console.log(q);//1
```

```
function f1() {
var x = 1
function f2() {
    var x = 2
    console.log(x);//2
}
f2();
console.log(x);//1
}
```

INSTITUTO FEDERAL
Espírito Santo

Nesse caso não ocorre sombreamento!

```
function f3() {
var x = "ola";
if (true) {
    var x = "mudei";
}
console.log(x);//mudei
}
```

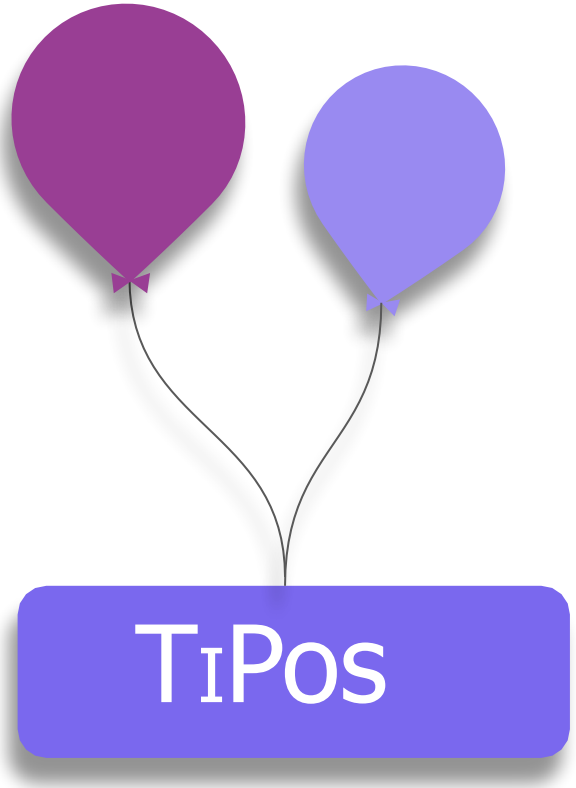
Com let o sombreamento é totalmente possível, não há múltiplas declarações.

```
function f3() {
    let sombra = "ainda sou eu";
    if (true) {
        let sombra = "não sou mais eu";
        console.log(sombra);//não sou mais eu
    }
    console.log(sombra);//ainda sou eu
}
```

- **Escopo**

TypeScript possui escopo estático e aninhado (Closures).

```
function f1() {
    let x = 1;
    function f2()
    {
        return x *= 10;
    }
    console.log(f2());
}
f1();//10
```



Compostos Vs Primitivos

INSTITUTO FEDERAL
Espírito Santo

- **Primitivos**
  - number: Representa valores de ponto flutuante IEEE 754 de precisão dupla de 64 bits.
  - boolean: Valores true e false.
  - string: Sequência de caracteres armazenados como unidades de código Unicode UTF-16
  - symbol: Representa tokens exclusivos que podem ser usados como chaves para propriedades de objetos.

- void: Representa a ausência de valor e é usado como o tipo de retorno de funções sem valor para retorno. Valores possíveis: undefined ou null.

- null: Faz referência a o primeiro e único valor do tipo Null.

- undefined: Denota o valor dado a todas variáveis não inicializadas.

- enum: São subtipos definidos pelo usuário do tipo primitivo number.

- literais: São tipos de valores exatos.

Exemplos

```
var aux: number = 100;  var
aux1: boolean = true;  var
aux2: string = "Yooh";
```

```
var nome:symbol = Symbol();
var obj = {};
obj[nome] = "Alguém";
console.log(obj[nome]);//Alguém
```

```
var aux5: void = null;
var aux6: null = null;
var aux7: undefined = undefined;
enum aux8 { Red, Blue, Green,};
```

```
type newType = "Norte" | "Sul"
| 10 | true;
var aux9: newTipe = 10;//Okay
aux9 = "Norte";//Okay
aux9 = "X"; //ERROR
```

INSTITUTO FEDERAL
Espírito Santo

- **Compostos**
  - Array: Lista de elementos do mesmo tipo.
  - Tupla: Tuplas são listas fixas de elementos heterogêneos.
  - Objeto: É qualquer coisa que não seja um tipo primitivo, (pode ser anônimo).
  - União: Pode assumir valores do conjunto de tipos adotados.
  - Interseção: Conjunto dos valores em comum.
  - Mapeado: Derivar um tipo de objeto em outro.

Exemplos

```
let list: number[] = [1, 2,   3];

let tupla: [number, number]   = [1.0, 2.0];

var objeto = {
    nome: "nome",
    idade: 10,
}
var x: string | number;
x = 10;//okay
x = "poh"//okay
x = true;//error
```

```
type tipo1 = string | number | boolean;
type tipo2 = string | boolean;
var t: tipo1 & tipo2;
t = "oi";//okay
t = true;//okay  t
= 1;//error
```

```
type x = 'a' | 'b' |
'c'; type map = { [keys
in x]: string };

let v: map = { a: "a",
b:
"b", c: "c" };
console.log(v.a);//a
```

- **Tipos especiais**
  - any: Assumem qualquer valor de tipos possíveis e desativa a checagem estática.
  - never: Representa os valores que nunca ocorrem, normalmente usado para funções que sempre retornam erro.

```
let aux10: any;
aux10 = 20;//okay!
aux10 = "eeeeba";//okay!
aux10 = true;//okay!
aux10 = {
nome: "Vitor",
idade: 30
};//okay!
```

```
let aux11: never;
aux11 = 1;//erro
aux11 = "s";//erro
aux11 = true;//erro
aux11 = null;//erro
aux11 = undefined; //erro
function f(): never {
    throw new Error("Útil?");
}//okay!
```

INSTITUTO FEDERAL
Espírito Santo

- **Persistência de dados**
  - Necessário importar como módulo a biblioteca fs do node.js
    - *readFile (path,options,callback) e writeFile(path,contents,options,callback)*
    - *readFileSync(path,options) e writeFileSync(path,contents,options)*

- **readFile e writeFile**

```
import * as fs from 'fs';
fs.writeFile("cof.txt", "GRRRRRRRRR", { 'flag': 'w' }, function
(err) {
    if (err) throw err
    console.log("success");
});
fs.readFile("cof.txt", { 'flag': 'r' }, function (err, data) {
    if (err) throw err;
    console.log(data.toString());//GRRRRRRRRR
})
```

- **readFileSync writeFileSync**

```
fs.writeFileSync("cof.txt",{ 'flag': 'w' },"au au
au");  var file = fs.readFileSync("cof.txt");
console.log(file.toString());//au au au
fs.writeFileSync("cof.txt", "\nmiau miau ", { 'flag': 'a+'
});  file = fs.readFileSync("cof.txt");
console.log(file.toString());//au au au\nmiau miau
```

- **Coletor de Lixo**
  - Marcar e varrer
  - Contagem de referência
- **Serialização**
  - JSON

    O TypeScript herda do JavaScript as serializações realizadas pelo JSON, existem plugins da comunidade que facilitam esse processo.

```
class Cachorro {
    name: string =
    "Dog-da-ufes";  idade:
    number = 100;  getName() {
        return this.name;
    }
    static fromJson(d: Object): Cachorro {
        return Object.assign(new Cachorro,
        d);
    }
}
```

```
var obj = new Cachorro;
var serializado = JSON.stringify(obj);
var obj_dnv = Cachorro.fromJson(JSON.parse(serializado));
console.log(serializado);//{"name":"Dog-da-ufes","idade":10
0}  console.log(obj_dnv.getName());//Dog-da-ufes
```

- **Operadores**
  - Operadores unários
  - Operadores binários
  - Operadores ternários
  - O operador this
  - O operador Spread

- Operadores unário

Operadores Aritméticos	
++	Incremental, prefixo ou posfixo, semelhante a C.
--	Decremental, prefixo ou posfixo, semelhante a C.

Operadores Bit a Bit	
~	Inverte a cadeia de bits.

Operadores especiais	
+	Converte qualquer tipo em number, positivo.
-	Converte qualquer tipo em number, negativo.
!	Transforma qualquer valor em booleano, pode ser combinado consigo mesmo várias vezes. Nega valores lógicos.
typeof	Pega um operando qualquer e gera o tipo do operando em forma de string.
delete	Deleta um elemento de um array ou propriedade de um objeto, se bem sucedido retorna true, senão, false.

Exemplos

```
let aux = 10;
console.log(aux++)//10
console.log(++aux)//12
console.log(aux--)//12
console.log(--aux)//10
```

```
let x = 10;
let v = '100';
let b = true;
console.log(+x);//10
console.log(+v);//10
0
console.log(+b);//1
```

INSTITUTO FEDERAL
Espírito Santo

```
let x = 10;
let v = '100';
let b = true;
console.log(-x);//-10
console.log(-v);//-10
0
console.log(-b);//-1
```

```
let x = 10;
let v = '100';
let b = true
console.log(~x);//-11
console.log(~v);//-10
1
console.log(~b);//-2
```

INSTITUTO FEDERAL
Espírito Santo

```
let x = 10;
let v = '100';
let b = true
console.log(!x);//false
console.log(!!v);//true
console.log(!!!b);//false
```

```
let x = 10;
console.log(typeof x);//number
let y: typeof x = "";//erro
var array = [1, 2, 3, 4]
console.log(delete  array[1]);//true
console.log(array);//[1,undefined,
3,4]
```

INSTITUTO FEDERAL
Espírito Santo

- Operadores Binários

Operadores Aritméticos	
*	Multiplicação
/	Divisão
%	Módulo
-	Subtração
**	Exponenciação
+	Soma

Operadores Bit a Bit	
^	Retorna 1 para cada posição de bit que um dos operandos possui 1 e o outro 0.
&	Retorna 1 em cada posição de bit para qual ambos operandos tem 1.
|	Retorna 1 para cada posição de bit quando ao menos um dos operandos tem 1.
<<	Desloca x quantidade de bits à esquerda, mandando 0s à direita.
>>	Descola x quantidade de bits à direita, descartando os bits que se tornam off.

>>>	Descola x quantidade de bits à direita, descartando os bits que se tornam off, e mandando os 0s a esquerda.

Operadores de Comparação	
>	Retorna true se o operando à esquerda for maior que o da direita, senão, retorna false.
<	Retorna true se o operando à esquerda for menor que o da direita, senão, retorna false.
<=	Retorna true se o operando à esquerda for menor ou igual ao da direita, senão, retorna false.

>=	Retorna true se o operando à esquerda for maior ou igual ao da direita, senão, retorna false.
==	Retorna true se o operando da esquerda for igual da direita, senão, retorna false.
!=	Retorna true se o operando da esquerda for diferente do da direita, senão, retorna false.
===	Retorna true se o operando da esquerda for igual em valor e em tipo ao da direita, senão, retorna false.
!==	Retorna true se o operando à esquerda for diferente em valor e/ou diferente em tipo, senão, retorna false.

Operadores Lógicos	
&&	Se os operandos forem expressões, retorna a expressão da esquerda se essa for possível converter para false, senão, a da direita. Caso os operandos sejam valores booleanos, retorna true se ambos forem true, senão, false.
||	Se os operandos forem expressões, retorna a expressão da esquerda se essa for possível converter para true, senão, retorna a direita. Caso os operandos sejam valores booleanos, retorna true se ao menos um for true, senão, false.

Operadores Relacionais	
in	Retorna true se a propriedade está presente no objeto.
instanceof	Retorna true se o objeto especificado a esquerda for do tipo do objeto a direita.

Operadores de Atribuição	
=	Atribui o valor a direita ao elemento da esquerda.
+=	Atribui o valor da direita somado com o valor do elemento a esquerda, ao elemento a esquerda.

-=	Atribui o valor da direita subtraído pelo valor do elemento a esquerda, ao elemento a esquerda.
*=	Atribui o valor da direita multiplicado pelo valor do elemento a esquerda, ao elemento a esquerda.
/=	Atribui o valor do elemento a esquerda dividido pelo valor da direita, ao elemento a esquerda.
%=	Atribui o resto da divisão do valor do elemento a esquerda pelo da direita, ao elemento da esquerda.
**=	Atribui o valor do elemento a esquerda elevado o valor da direita, ao elemento da esquerda.

<<=	Atribui o valor do elemento a esquerda deslocado o valor da direita bits para a esquerda, ao elemento da direita.
>>=	Atribui o valor do elemento a esquerda deslocado o valor da direita bits para a direita, ao elemento da direita.
>>>=	Atribui o valor do elemento a esquerda deslocado o valor da direita bits para a direita, ao elemento da direita. De forma não assinada.
&=	Atribui a operação & bit a bit do elemento a esquerda com o da direita.
^=	Atribui a operação ^ bit a bit do elemento a esquerda com o da direita.

|=	Atribui a operação | bit a bit do elemento a esquerda com o da direita.

Exemplos

```
var x = 10;  var y = 2;  x ^= y;

console.log(x);//8
```

```
x <<= y;

console.log(x);//32  x >>>= y;

console.log(x);//8
```

- Operador Ternário

Operadores Condicionais	
exp ? retorno 1 : retorno 2	Questiona se a exp pode ser atribuída como true, se sim, retorna o retorno 1, senão, o retorno 2.

```
var a = { name: "ciclano", idade: -10 };  var
b = a.idade ? a : null;
console.log(b);//{name:'ciclano',idade:-10}
```

- O operador this

Em TypeScript o this continua sendo uma palavra chave de referência a um objeto, só que com algumas armadilhas de escopo.

```
var a = {
    name: "ciclano", idade: -10, imprime()
        {  console.log(this.name,
        this.idade);
};  }       a.imprime();//ciclano
            -10  var p = a.imprime;
            p();//undefined
            undefined
```

```
var a = {
    name: "ciclano", idade: -10,
 imprime() {
        console.log(this.name, this.idade);
        function imprime2() {
            console.log(this.name, this.idade);//erro
        }
        imprime2();
    }
};
```

```
var a = {
    name: "ciclano", idade: -10, imprime() {
        console.log(this.name, this.idade);  var _this =
        this;
        function imprime2() {  console.log(_this.name,
            _this.idade);
        }
        imprime2();
    }
};
      a.imprime();// ciclano -10\n ciclano -10
```

INSTITUTO FEDERAL
Espírito Santo

```
var a = {
    name: "ciclano", idade: -10, imprime() {
        console.log(this.name, this.idade);  var b = () => {
            console.log(this.name, this.idade);
        }  b();

    }
};


a.imprime();// ciclano -10\n ciclano -10
```

- O operador Spread (...)

  Permite a expansão de uma expressão em locais onde se esperam vários argumentos.

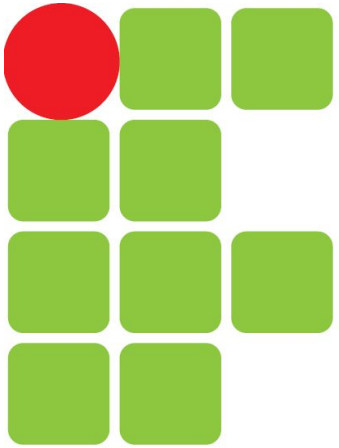
```
let array1 = [1, 2, 3]
let array2 = [-1, -2, 0, ...array1, 4, 5,
6]
console.log(array2)//[-1,-2,0,1,2,3,4,5,6]
```

Referências

TypeScript is JavaScript with syntax for types. https://www.typescriptlang.org

Guia Pratico de Typescript https://github.com/KAYOKG/BibliotecaDev/blob/main/LivrosDev/Guia%20pr%C3%A1tico%20de%20TypeScript%20-%20Melhore%20suas%20aplica%C3%A7%C3%B5es%20JavaScript%20-%20Autor%20(Casa%20do%20C%C3%B3digo).pdf

TypeScript TypeScript is a typed superset of JavaScript that compiles to plain JavaScript. https://devfreebooks.github.io/typescript/



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PERNAMBUCO