

# Seminário 1

## Aspectos do TypeScript

Alexandre Júlio Lima Mendes

Instituto Federal do Norte de Minas Gerais  
Campus Montes Claros

18 de abril de 2023



# Índice

## 1 Intro

- História
- Características gerais

## 2 Avaliação

- Como avaliar uma linguagem
- Simplicidade
- Ortogonalidade
- Avaliações adicionais

## 3 Variáveis

- Declarações com var
- Declarações com let
- var vs let
- Declarações com const

## 4 Vinculação

- Tipagem Estática/Dinâmica
- Tipagem forte/fraca

## ■ Asserção de tipo

## 5 Tipos

- Tipo number
- Tipo string
- Operadores
- Tipo ordinal (enum)

## 6 Regex

## 7 Array/Matriz

- Arrays
- Matrizes
- Matrizes associativas (tuplas)

## 8 Registros

- Union
- Classes
- Interface

## 9 Conclusão



# Intro



# Introdução

## Origem

- Lançada para uso público em 2012
- Anders Hejlsberg, principal arquiteto do C#, Delphi e Turbo Pascal
- Criado para lidar com desenvolvimento de programas grandes
- Os desafios de lidar com código JavaScript grande e complexo
- Demanda por ferramentas personalizadas para facilitar o desenvolvimento de componentes na linguagem

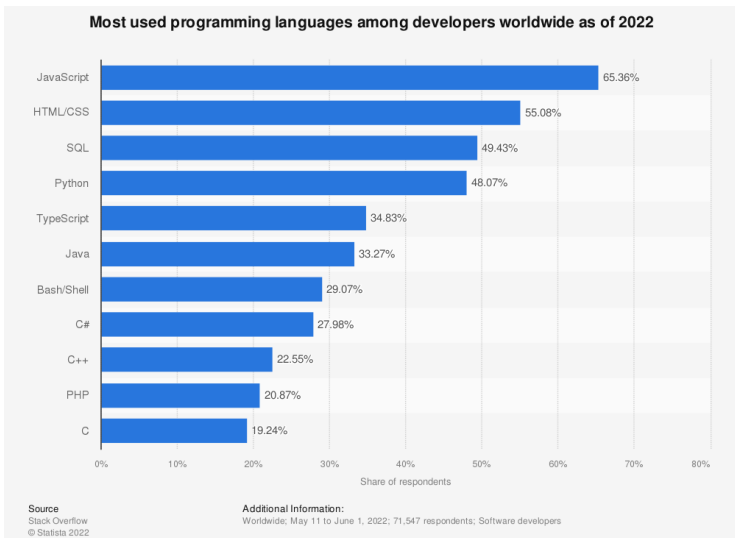


## Características gerais

- Desenvolvida e mantida pela microsoft
- Open source
- Superconjunto Javascript, pode ser transcompilado para este
- Todos programas Javascript podem ser considerados Typescript



# Uso e popularidade



# Uso e popularidade

## Most Loved, Dreaded, and Wanted Languages

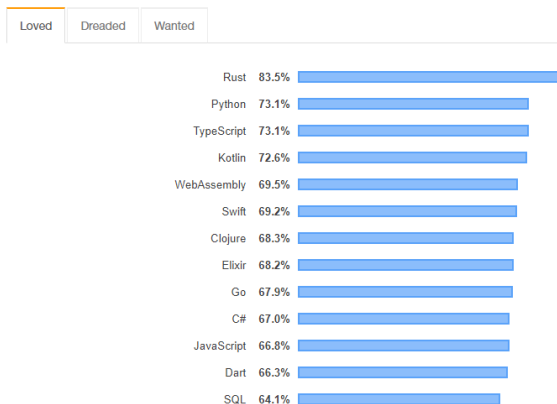


Figura: % de desenvolvedores que estão desenvolvendo com a linguagem ou tecnologia e manifestaram interesse em continuar desenvolvendo com ela



# Avaliação





# Critérios de Avaliação

- Legibilidade
  - Simplicidade Geral
  - Ortogonalidade
  - Tipos de dados
  - Projeto da Sintaxe
- Facilidade da escrita
  - Simplicidade e ortogonalidade
  - Suporte à abstração
  - Expressividade
- Confiabilidade
  - Verificação de tipos
  - Tratamento de exceções
  - Apelidos (apontadores/ponteiros)
- Custo
- Portabilidade
- Generalidade



# Simplicidade



```
const name: string = 'John'
```

- Por ser subconjunto do Javascript, possui construções adicionais, adicionando complexidade
- Linguagens voltadas a web possuem complexidade adicional. Atrrelaçadas com HTML/CSS
- Muitas construções
- Multiplicidade de recursos, possível realizar um mesmo objetivo de várias formas e construções diferentes



# Construções

## Recursos Adicionais

- Anotações de tipo e verificação de tipo em tempo de compilação
- Inferência de tipo
- Interfaces
- Tipos enumerados
- Genéricos
- Namespaces
- Tuplas
- Async/await

## Recursos Presentes no Javascript


- Classes
- Módulos
- Sintaxe abreviada de "seta" para funções "anônimas"
- Parâmetros opcionais e parâmetros padrão



# Ortogonalidade

## O tipo Void

- Similar a outras linguagens - se uma função não retorna dados, ela é do tipo void
- Usado quando não há dados
- É possível declarar variáveis do tipo void
- Podem receber apenas valores "NULL" ou "undefined", mostrando falta de ortogonalidade



```
let nothing: void = 'undefined'  
let num: void = 1 //Erro
```



# Avaliações adicionais

## Custo

- Usar Typescript pode reduzir o custo de aplicações web, tanto por ser mais fácil/rápido de treinar desenvolvedores que já sabem Javascript(que já é imensamente popular) e por ser uma linguagem mais previsível
- A Airbnb relatou em 2019 uma redução de 38% em bugs ao adicionar Typescript no processo de desenvolvimento

## Legibilidade

- Algumas características de tipagem forte (tipos estritos) torna o código mais autoexpressivo

## Generalidade e Portabilidade

- Todo dispositivo, plataforma ou navegador que executa Javascript também funciona com Typescript.
- Javascript é implementado consistentemente por navegadores, o tornando Javascript e Typescript muito portáveis. Para executar, só é necessário um navegador



# Variáveis



# Declarações com var

## var

- Declarações de variáveis, no Javascript, eram tradicionalmente feitas com a *keyword* **var**
- Possui algumas peculiaridades




```
//var [identificador]: [tipo] = valor;  
var nome: string = 'Pedro' // A variável armazena o valor de tipo string  
  
//var [identificador]: [tipo];  
var nome: string // A variável é tipo string. O valor é undefined por padrão  
  
//var [identificador] = valor;  
var nome = 'Pedro' // O tipo é inferido pelo valor. Nesse exemplo, é string  
  
//var [identificador];  
var name // O tipo é any. O valor é undefined por padrão
```



# Regras de escopo com var

- Variáveis declaradas com var são acessíveis em qualquer lugar dentro da função que as contém.



```
function f(entraNoIf: boolean) {  
  if (entraNoIf) {  
    var x = 10  
  }  
  return x  
}  
f(true) // retorna '10'  
f(false) // retorna 'undefined'
```

- Neste exemplo, x foi declarada dentro do if.
- No entanto, mesmo se não entrarmos no if, x ainda é declarado.
- Se não entrarmos no if, x é declarado, mas **não** inicializado.





# Declarações com let

```
function f(input: boolean){  
  let a = 100;  
  if(input) {  
    //OK. 'a' pode ser referenciado aqui.  
    let b = a + 1;  
    return b;  
  }  
  //Erro: 'b' não existe aqui.  
  return b;  
}
```

- Variáveis possuem "escopo de bloco".
- Não são visíveis fora do escopo no qual foram declaradas.
- A variável 'b' foi declarada dentro do escopo do if. Portanto, ela não existe fora deste.



# var vs let

## Mesmo nome e sensibilidade a capitalização

- Variáveis declaradas com let:
  - Não podem ser declaradas com o mesmo nome
  - Não são sensíveis a capitalização
- Variáveis declaradas com var:
  - Podem ser declaradas com o mesmo nome

```
var num:number = 1; // OK
var num:number = 2; //OK
var Num:number = 3; // OK
var NUM:number = 4; // OK
var NuM:number = 5; // OK
var num:number = 6; //OK

let num:number = 7; // Compiler Error: Cannot redeclared block-scoped variable 'num'
let Num:number = 8; // Compiler Error: Cannot redeclared block-scoped variable 'Num'
let NUM:number = 9; // Compiler Error: Cannot redeclared block-scoped variable 'NUM'
let NuM:number = 10; // Compiler Error: Cannot redeclared block-scoped variable 'NuM'
```



# var vs let

## var hoisting

- Variáveis declaradas com var são "*hoisted*" ou "*içadas*" para cima
- São definidas quando a função começa, independente de onde são declaradas
- Uso de var pode ser imprevisível, recomendado utilizar o let
- Falta de ortogonalidade na declaração de variáveis

```
function digaOi() {  
  var frase;  
  frase = 'Oi';  
  alert(frase); //imprime  
}  
digaOi();
```

```
function digaOi() {  
  frase = 'Oi';  
  
  if(false){  
    var frase;  
  }  
  
  alert(frase); //imprime  
}  
digaOi();
```

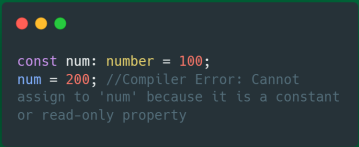
```
function digaOi() {  
  frase = 'Oi';  
  alert(frase); //imprime  
  var frase;  
}  
digaOi();
```

Figura: Nos três exemplos, será impresso 'Oi' sem erros



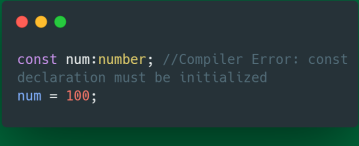
# Declarações com const

- Declarações de variáveis com const possuem as mesmas regras de escopo de let
- Variáveis declaradas com const não podem ter seu valor mudado.



```
const num: number = 100;  
num = 200; //Compiler Error: Cannot  
assign to 'num' because it is a constant  
or read-only property
```

- Variáveis constantes devem ser declaradas e inicializadas em uma única expressão.



```
const num:number; //Compiler Error: const  
declaration must be initialized  
num = 100;
```



# const Object

- Variáveis declaradas com const permitem que propriedades de um objeto sejam modificadas, mas não permitem modificação da estrutura do objeto



```
const playerCodes = {  
  player1 : 9,  
  player2 : 10,  
  player3 : 13,  
  player4 : 20  
};  
playerCodes.player2 = 11; // OK  
  
playerCodes = {           //Compiler Error:  
  Cannot assign to playerCodes because it  
  is a constant or read-only  
  player1 : 50,           // Modified value  
  player2 : 10,  
  player3 : 13,  
  player4 : 20  
};
```



## Vinculação



# Tipagem

## Dinâmica

- Permite que a variável seja declarada sem determinar o tipo se não for atribuído nenhum valor
- Se for atribuído valor, é observado o tipo do valor sendo atribuído e isso define o tipo
- É possível informar explicitamente o tipo, mas não é obrigatório



## Características de tipagem forte e fraca

- Possui características de tipagem fraca, como por exemplo, entre operações aritméticas entre números e strings
- Também possui características de tipagem forte, por exemplo na divisão de um número por um array

```
var num = 5, str = "4";  
var result = num + str; //retorna "54"
```

```
console.log(4 / []);  
// Erro de compilação em Typescript.  
Compila e executa no Javascript.
```





# Asserções de tipo

- Se estiver sendo usado `document.getElementById`, o Typescript sabe apenas que isso retornará um `HTMLElement`
- Mas **você** pode saber que sua página sempre terá um `HTMLCanvasElement` com um determinado ID
- Pode ser usado uma declaração de tipo para especificar um tipo mais específico



```
const myCanvas = document.getElementById("main_canvas") as HTMLCanvasElement;
```



# Tipos



# Tipos

## O Typescript possui vários tipos primitivos

- number
- string
- boolean
- void
- any

## Também possui outros tipos, que podem conter tipos primitivos

- arrays
- tuplas
- union
- interfaces



# Tipo number

- Variáveis do tipo number são de ponto flutuante. Existe também o bigint.
- Possuem métodos. Ex.: `toString()`, `toExponential()`, `toPrecision()`

```
let decimal: number = 6;  
let hex: number = 0xf00d;  
let binary: number = 0b1010;  
let octal: number = 0o744;  
let big: bigint = 100n;
```

## Acurácia

- Até 15 dígitos para inteiros
- Até 17 dígitos para números de ponto flutuante



# Tipo string

- String é outro tipo primitivo usado para armazenar dados em texto
- Possui tamanho dinâmico
- Também possui métodos. Ex.: `charAt()`, `concat()`, `split()`, `endsWith()`

```
let nomeFuncionario:string = 'John Smith';  
//ou let nomeFuncionario:string = "John Smith";  
  
let descricaoFuncionario: string = nomeFuncionario + " trabalha no departamento de " + nomeDepartamento + ".";  
  
let descricaoFuncionario2: string = `${nomeFuncionario} works in the ${nomeDepartamento} department.`;  
//Template String
```



# Operações entre tipos number e string

- Operador de adição + concatena e retorna uma string se qualquer um dos dois valores for string

```
var num = 5, str = "4";  
var result = num + str; //retorna "54"
```

- Outras operações aritméticas sobre strings numéricas resultam em números

```
var numStr1 = "5", numStr2 = "4";  
  
var multiplication = numStr1 * numStr2;  
//retorna 20  
var division = numStr1 / numStr2;  
//retorna 1.25  
var modulus = numStr1 % numStr2;  
//retorna 1
```



# Tipo ordinal (enum)

## Tipos de enum

- Numérico
- String
- Heterogêneo

```
enum PrintMedia {
    Newspaper, // 0
    Newsletter, // 1
    Magazine, // 2
    Book // 3
}
//também é possível inicializar o primeiro valor
numérico. Ex.:
//Newspaper = 1,
```

```
enum PrintMedia {
    Newspaper = "NEWSPAPER",
    Newsletter = "NEWSLETTER",
    Magazine = "MAGAZINE",
    Book = "BOOK"
}
//acessar o string enum
PrintMedia.Newspaper; //retorna NEWSPAPER
PrintMedia['Magazine'];//retorna MAGAZINE
```




# Regex





# Expressões regulares



```
let sampleRegexMail =  
new RegExp( '^[a-z0-9._%+-]+@[a-z0-9-]+\.[a-z]{2,4}$' );  
console.log( sampleRegexMail.test( 'educba45@sample.com' ) )  
// result: true  
console.log( sampleRegexMail.test( 'educba.sample@.com' ) )  
// result: false  
console.log( sampleRegexMail.test( 'educba_@sample23.inn' ) )  
// result: true  
console.log( sampleRegexMail.test( 'educbasample$%^com' ) )  
// result: false  
console.log( sampleRegexMail.test( '2354@2324.com' ) )  
// result: true
```



# Array/Matriz



# Arrays

```
let fruits: Array<string>;
fruits = ['Apple', 'Orange', 'Banana'];

let ids: Array<number>;
ids = [23, 34, 100, 124, 44];

let values: (string | number)[] = ['Apple', 2, 'Orange', 3, 4, 'Banana'];
// ou let values: Array<string | number> = ['Apple', 2, 'Orange', 3, 4, 'Banana'];

let fruits: string[] = ['Apple', 'Orange', 'Banana'];

for(var index in fruits)
{
    console.log(fruits[index]); // output: Apple Orange Banana
}

for(var i = 0; i < fruits.length; i++)
{
    console.log(fruits[i]); // output: Apple Orange Banana
}
```



# Matrizes



```
var matriz:number[][] = [[1,2,3],[23,24,25]] ;  
  
const numeroFuncionario:[number][string] = [[10,'A'],[20,'B'],[30,'C']];
```

## Exemplos de métodos de Arrays e Matrizes

- pop(), push(), sort(), slice(), lastIndexOf(), toString(), indexOf()



# Matrizes associativas (tuplas)



```
//tuplas são arrays comuns  
var mytuple = [10,"Hello"]; //cria uma tupla  
console.log(mytuple[0]) //10  
console.log(mytuple[1]) //Hello
```



# Registros



# Union



```
let code: (string | number);  
code = 123; // OK  
code = "ABC"; // OK  
code = false; // Compiler Error
```

```
let empId: string | number;  
empId = 111; // OK  
empId = "E111"; // OK  
empId = true; // Compiler Error
```



# Classes

```
class Funcionario {  
    codigo: number;  
    nome: string;  
  
    constructor(cod: number, name: string) {  
        this.nome = name;  
        this.codigo = cod;  
    }  
  
    getSalario() : number {  
        return 10000;  
    }  
}
```





# Interface

- Interface é uma estrutura que define um contrato em uma aplicação
- Define a sintaxe para as classes a seguir
- Classes derivadas de uma interface devem seguir a estrutura fornecida por sua interface

```
interface Pessoa {  
  nome: string;  
  genero: string;  
}  
  
interface Funcionario extends Pessoa {  
  codigo: number;  
}  
  
let funcionarioObj:Funcionario = {  
  codigo:1,  
  name:"Bill",  
  gender:"Masculino"  
}
```



# Conclusão



# Conclusão

## Pontos adicionais

- Não permite o uso de apontadores
  - Melhor confiabilidade
- Possui coletor de lixo
  - Não existem deletes. Estruturas dinâmicas como strings precisam ser desalocadas. Isso acontece pelo coletor de lixo da linguagem
  - Clique [aqui](#) e [aqui](#) para leitura adicional sobre gerenciamento de memória na linguagem

## Avaliação

- A popularidade do typescript é justificada
- Se apresenta como uma linguagem altamente modular, flexível e confiável se forem utilizadas suas construções
- Melhoria significativa do javascript quando considerados projetos maiores



Perguntas?

