

VEM SER

TypeScript

Aula 01 - Introdução



Tipagem
de
dados

O que é tipagem?

Tipagem de dados são as categorias em que as linguagens de programação distinguem os dados que são armazenados nas variáveis.

Os tipos comuns são:

- **Inteiros** // *int*
- **Reais** (ou flutuantes) // *float/double*
- **Booleanos** // *boolean*
- **Textos** // *string*

Tipagem do JavaScript

Os tipos comuns no JavaScript são:

- ***Number*** (inteiros e flutuantes);
- ***Booleano***;
- ***String***;
- ***Null***.



Tipagem
estática
vs
dinâmica

O que é tipagem estática?

Numa linguagem de tipagem estática há a obrigatoriedade de **declarar o tipo de dado que será armazenado na variável**, além do valor.

Também devem ser declarados os tipos dos parâmetros nas funções e o tipo do retorno.

Exemplos de linguagem que são estaticamente tipadas: C, C++, C#, Java, Go, Rust



The screenshot displays the IntelliJ IDEA IDE. The top pane shows the source code of `Main.java` with line numbers 1 through 7. The code defines a `Main` class with a `main` method. In line 3, the variable `numero` is declared as an `int` and assigned the value 1. The `main` method prints the value of `numero` and its runtime class name. The bottom pane shows the execution output for the `Main` class. The command used to run the program is `/opt/jdk-19.0.1/bin/java -javaagent:/snap/intellij-idea-ultimate/386/lib/...`. The output shows the value `1` and the class name `Integer`, both of which are highlighted with red boxes. The process finished with exit code 0.

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int numero = 1;  
4         System.out.println(numero);  
5         System.out.println(((Object)numero).getClass().getSimpleName());  
6     }  
7 }
```

Run: Main x

`/opt/jdk-19.0.1/bin/java -javaagent:/snap/intellij-idea-ultimate/386/lib/...`

1

Integer

Process finished with exit code 0

```
main.c x + >_ Console x
> f main
1 #define typeof(var) _Generic( (var),\
2 char: "Char",\
3 int: "Integer",\
4 float: "Float",\
5 char *: "String",\
6 void *: "Pointer",\
7 default: "Undefined")
8
9 int main(void) {
10     int numero = 1;
11     printf("%s\n", typeof(numero));
12     return 0;
13 }
```

```
> make -s
./main
Integer
```

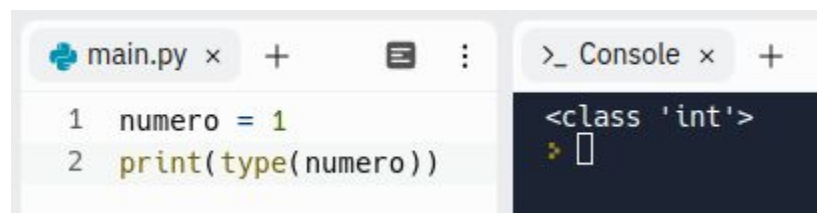

O que é tipagem dinâmica?

Uma linguagem é dita de tipagem dinâmica quando, durante a execução, infere qual o tipo de cada variável **sem necessidade de declarar o tipo do dado na inicialização da variável**. Vale ressaltar, também, que esse tipo pode mudar durante o decorrer do programa.

Exemplos de linguagens que são dinamicamente tipadas: JavaScript, Python, PHP, Ruby

Inferência de tipo

```
> let numero = 1  
console.log(typeof numero)  
number VM237:3
```



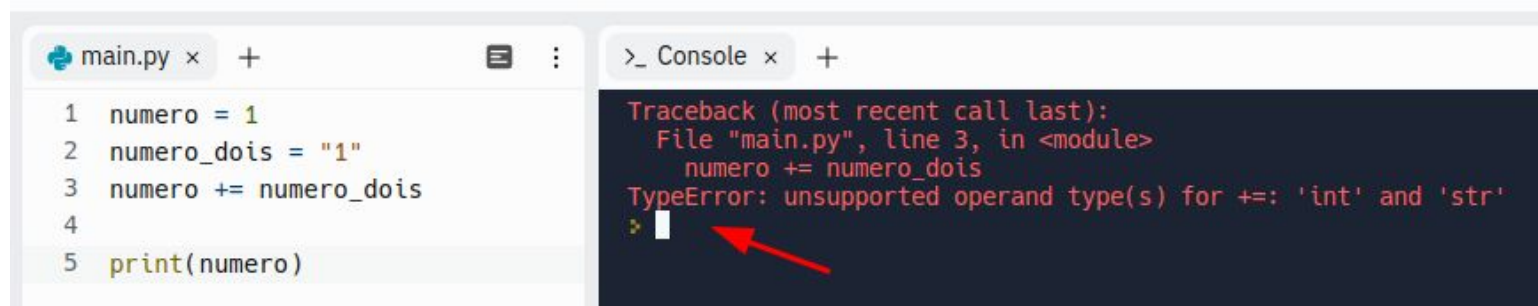
The screenshot shows a code editor with a file named 'main.py' containing two lines of Python code: '1 numero = 1' and '2 print(type(numero))'. To the right of the editor is a console window titled '>_ Console' which displays the output of the code: '<class 'int''>'. A cursor is visible at the end of the output line in the console.

```
main.py x + ⋮  
1 numero = 1  
2 print(type(numero))  
  
>_ Console x +  
<class 'int'>  
➤
```

Tipagem: Forte vs Fraca

Dizemos que uma linguagem é fortemente tipada quando, para alterar o tipo do dado da variável, é necessária uma **conversão direta e explícita**.

Exemplo de tipagem forte: Python



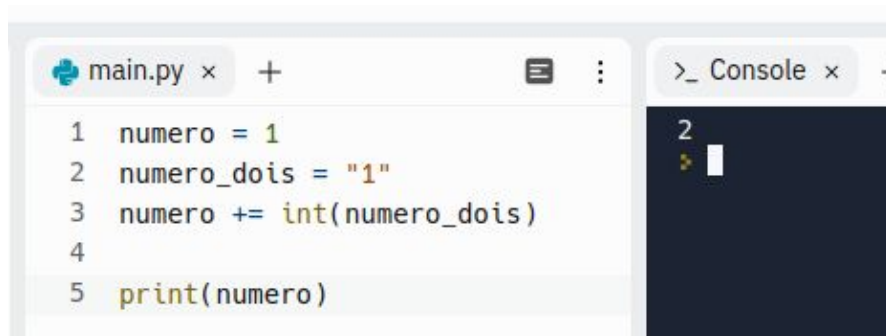
The screenshot shows a Python IDE with a file named `main.py` and a console window. The code in `main.py` is:

```
1 numero = 1
2 numero_dois = "1"
3 numero += numero_dois
4
5 print(numero)
```

The console window displays a `TypeError` message:

```
Traceback (most recent call last):
  File "main.py", line 3, in <module>
    numero += numero_dois
TypeError: unsupported operand type(s) for +=: 'int' and 'str'
```

A red arrow points to the `numero += numero_dois` line in the code editor, indicating the source of the error.



The screenshot shows the same Python IDE with the code in `main.py` modified to cast the string to an integer:

```
1 numero = 1
2 numero_dois = "1"
3 numero += int(numero_dois)
4
5 print(numero)
```

The console window now shows the output:

```
2
```

Exemplo de tipagem fraca: JavaScript

Reparem que não houve nem um erro ao tentar “somar” um número com uma string.

```
> numero = 1  
numeroDois = "1"  
  
console.log(numero+numeroDois)  
11
```

VM265:4



Qual a “mágica” do JavaScript?

Aconteceu o que chamamos de **conversão implícita (ou coerção implícita)**.
O JavaScript, internamente, “infere” um tipo que resultará da operação.

Tipagem...?

Uma linguagem pode ter tipagem dinâmica e ser fortemente tipada, como o Python.

Mas também pode ter **tipagem dinâmica** e ser **fracamente tipada**, como o **JavaScript**.
Uma coisa não influencia a outra.

Referências

<https://www.alura.com.br/artigos/o-que-sao-as-tipagens-estatica-e-dinamica-em-programacao>

<https://dev.to/joaoava/tipagem-frac-forte-dinamica-e-estatica-g8k>

<https://dicasdeprogramacao.com.br/tipos-de-dados-primitivos>





O que é o TypeScript?

Podemos brincar dizendo que o TypeScript, ou TS, é o “JavaScript com superpoderes”.

Em termos técnicos, o TS é um ***superset*** (conjunto de funcionalidades) acrescentadas à linguagem JavaScript. Trabalhamos em uma “camada” acima do JavaScript.

É “**como se fosse**” o JavaScript com **tipagem estática**.

No TS, para cada variável, parâmetro e retorno, precisamos especificar **o tipo do valor** armazenado.

Quais as vantagens?

- Detectar erros mais rápido durante o desenvolvimento;
- Detectar erros antes do código ir para o ambiente de produção;
- Funcionalidades que não estão disponíveis nativamente no JS;
- A orientação a objetos é melhor de se trabalhar no TS;
- Mais segurança no código no estágio de desenvolvimento;
- Menos *bugs* no estágio de produção;
- ...



Antes da prática...

TypeScript é uma linguagem que precisa de um **transpilador**.

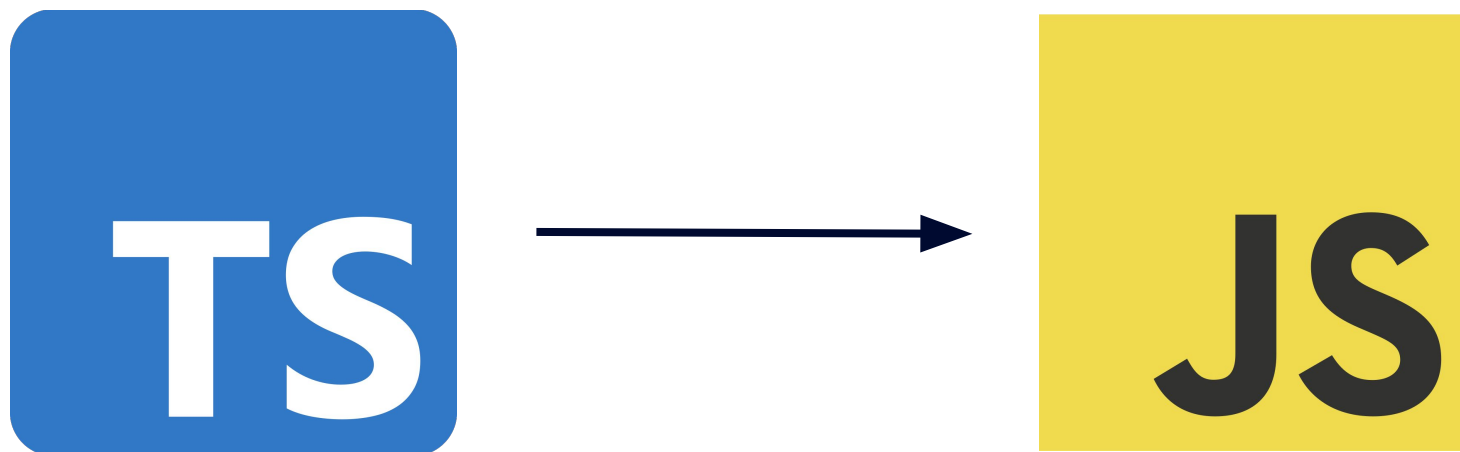
O que é um compilador?

É um programa que converte um arquivo de determinada linguagem para linguagem de máquina.



O que é um transpilador?

Enquanto o compilador converte para linguagem de máquina, um transpilador basicamente funciona como um “tradutor”. Ou seja, ele converte da linguagem X para a linguagem Y.





Instalando o ts-node

No terminal, digite:

```
npm i -g ts-node
```

Instalando o TSC

// TSC = TypeScript Compiler

No terminal, digite:

```
npm i -g typescript
```


Rodando um arquivo TS com ts-node

```
mayra in ~/Downloads  
→ ts-node teste.ts  
teste
```

Traduzindo TS para JS com o TSC

```
+  
mayra in ~/Downloads  
→ tsc teste.ts
```



Traduzindo em *realtime*

Na pasta em que está o arquivo TS abra o terminal (pode ser o Git Bash também) e digite:

tsc --init


tsc -w // ou também **tsc --watch**



Tipos
no
TypeScript

String

JS



```
1 let nome = "Mayra";
```

TS



```
1 let nome: string = "Mayra";
```

Number

JS



```
1 let numero = 10;
```

TS



```
1 let numero: number = 10;
```

Boolean

JS



```
1 let booleanJs = true;
```

TS



```
1 let booleanTs: boolean = true;
```

any

Usar em casos MUITO específicos.



```
1 let variavelQualquer: any = `Pode ser reatribuída como  
2 string, number, boolean, etc...`
```


Array de strings

Array em JS



```
1 let arrayDeStringsJs = ["JavaScript", "React", "TypeScript"];
```

Array em TS



```
1 let arrayDeStringsTs: string[] = ["JavaScript", "React", "TypeScript"];
```

Array de numbers

Array em JS



```
1 let arrayDeNumberJs = [0, 1, 2, 3];
```

Array em JS



```
1 let arrayDeNumberTs: number[] = [0, 1, 2, 3];
```

Tuplas

Basicamente é “como se fosse” um *array*, só que com vários tipos.



```
1 let tupla: [string, number, number] = ["Tv. São José", 455, 90240200];
```

Enum

Enum é um tipo que permite declarar um conjunto de constantes (ou valores) pré-definidos.

```
1 // usem Pascal Case
2 enum MesesNoDate {
3     Janeiro,
4     Fevereiro,
5     Março,
6     Abril,
7     Maio,
8     Junho,
9     Julho,
10    Agosto,
11    Setembro,
12    Outubro,
13    Novembro,
14    Dezembro
15 }
```

Enum


Criando uma variável **janeiro** com base no *Enum MesesNoDate*:



```
1 let janeiro: MesesNoDate = MesesNoDate.Janeiro;
```

Funções: parâmetros

Na função abaixo estamos recebendo um parâmetro **numero**, cujo tipo do dado é *number*.



```
1  function ehMaiorQueDez(numero: number) {  
2      if(numero > 10) {  
3          return true;  
4      } else {  
5          return false;  
6      }  
7  }
```

Funções: retorno

Na função abaixo estamos retornando uma *string*.



```
1  function retornaDbc(): string {  
2      return 'DBC Company';  
3  }
```

Funções sem retorno

Quando uma função não retorna nada, dizemos que é uma função *void*.



```
1  function printaAlgumaCoisa(): void {  
2      console.log('Vem Ser DBC');  
3  }
```


Funções: parâmetros e retorno



```
1  function ehMaiorQueDez(numero: number): boolean {  
2      if(numero > 10) {  
3          return true;  
4      } else {  
5          return false;  
6      }  
7  }
```

Objetos



```
1 let animal: { raza: string, nome: string, idade: number } = {
2   raza: 'dálmata',
3   nome: 'Luke',
4   idade: 5
5 }
```



Tipos
personalizados

Types



```
1 type Aluno = {  
2     instrutores: string[],  
3     modulosDasAulas: string[]  
4 }
```

Union Types

Quando uma variável pode ter um tipo **ou** outro, podemos usar as *union types* para definir os tipos aceitos.



```
1 let fezHomework: boolean | string;
2 // As duas atribuições serão aceitas:
3 fezHomework = 'sim';
4 fezHomework = true;
```

O tipo polêmico: *Null*

O tipo **nulo** é muito útil, principalmente para guardar informações no banco de dados, servindo como uma espécie de “*placeholder*”.

O tipo polêmico: *Null*



Créditos da imagem: @onebitcode

O tipo polêmico: *Null*

Vamos supor que temos um tipo personalizado chamado "CadastroUsuario", e o usuário pode ter um ou dois telefones cadastrados, sendo o primeiro obrigatório e o segundo opcional.



```
1 type CadastroUsuario = {  
2     nome: string[], // array de strings  
3     endereco: [string, number, number], // tupla  
4     telefoneUm: number,  
5     telefoneDois: number | null  
6 }
```


O tipo polêmico: *Null*

Caso o usuário cadastre apenas o primeiro telefone, o segundo terá o tipo *null*.

Porém, caso deseje editar seu cadastro algum dia e salvar um segundo número, também será possível, pois *telefoneDois* recebe dois tipos de valores: **números** ou o tipo **nulo** (*null*).



Classes no
TypeScript

Forma comum



```
1 class Turma {  
2     stack: string;  
3  
4     constructor(stack: string) {  
5         this.stack = stack;  
6     }  
7 }
```

Forma concisa



```
1 class TurmaConcisa {  
2     constructor(public stack: string) {}  
3 }
```

public? private?

...

A large, dark blue speech bubble with a white border, containing the text 'Modificadores de acesso'.

Modificadores de acesso

Modificadores de acesso

Os modificadores de acesso controlam a visibilidade dos atributos e métodos de uma classe. Existem três:

- ***public***: pode ser acessado em qualquer parte do código;
- ***protected***: pode ser acessado dentro da própria classe e por classes filhas (subclasses);
- ***private***: pode ser acessado somente dentro da própria classe.



Métodos

Métodos

Métodos são “comportamentos” de uma classe (ou da instância dela).
Imagine que temos uma classe Aniversário:

```
1 class Aniversario {  
2     constructor(public dia: number, public mes: number, public ano: number) {}  
3 }
```

Na prática


Vamos criar o construtor da classe:



```
1 class Aniversario {  
2     idade: number;  
3     dataAtual: Date = new Date();  
4  
5     constructor(public dia: number, public mes: number, public ano: number) {  
6         if (this.dataAtual.getMonth() ≥ this.mes - 1) {  
7             this.idade = this.dataAtual.getFullYear() - this.ano;  
8         } else {  
9             this.idade = this.dataAtual.getFullYear() - this.ano - 1;  
10        }  
11    }  
12 }
```

Na prática


Vamos criar o método "fezAniversario":



```
1  fezAniversario(): string {  
2      this.idade++;  
3      return `Parabéns! Agora você tem ${this.idade} anos!`;  
4  }
```

Na prática

E, como idade é privada, vamos criar o método “verIdade”:



```
1  verIdade(): number {  
2    return this.idade;  
3  }
```



Recomendações de leitura

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



Let's *Tech Up Together*