

***Fabiano Moreira***

**professor@fabianomoreira.com.br**

# O que é Angular



**Framework para construir aplicações web baseadas em HTML5, CSS e Javascript**

**Organiza essas tecnologias e entrega uma aplicação que executa no browser, que pode consumir um ou vários serviços disponibilizados por um servidor.**

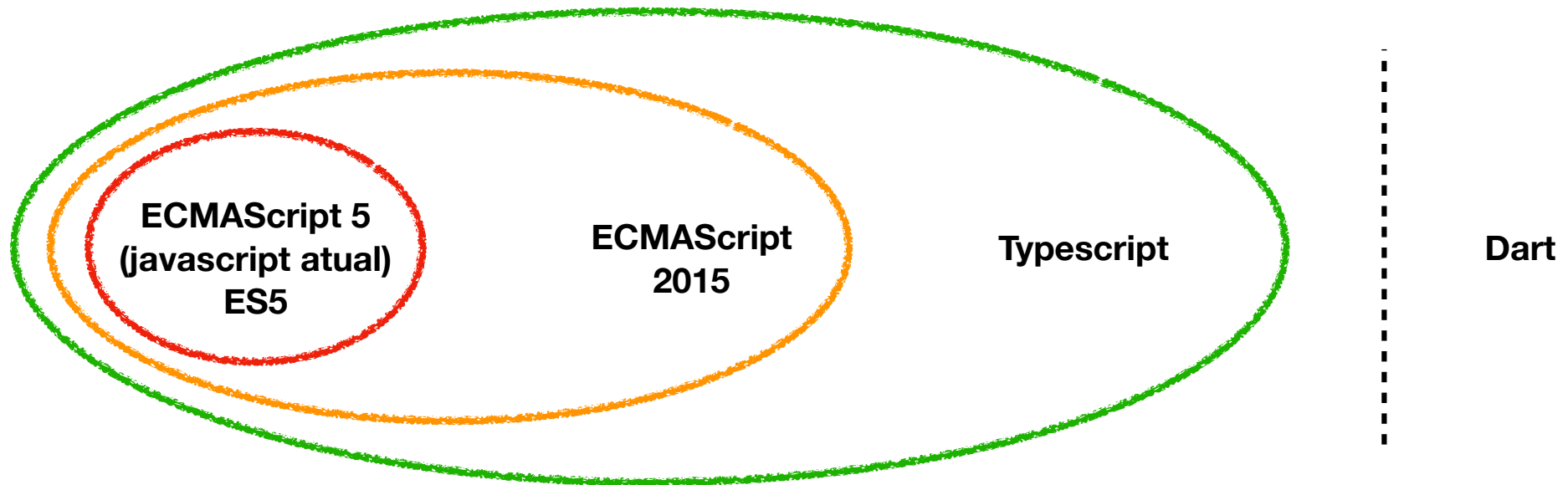


**Aplicações **client side****

# Linguagem



Existem 4 linguagens que podem ser usadas para desenvolver com angular



- Não precisa de compilador
- API confusa
- Não suporta tipos

- Classes
- Módulos
- Precisa de um compilador
- Não suporta tipos

- Linguagem usada para criar o angular
- Estende o ECMA 2015
- Possui tipos, classes, interfaces, módulos, decorators
- Possui um compilador

# Ambiente de desenvolvimento



## node.js

- Plataforma criada a partir do google chrome
- Usado para aplicações server side
- Gerenciamento das dependências da aplicação através do npm (Node Package Manager)

<https://nodejs.org/en/>

No terminal: `node -v`

## Angular cli

- Ferramenta de linha de comando
- Cria desde o projeto inicial até a preparação da aplicação para produção
- Cria os componentes, diretivas, pipes, etc...

`npm install -g @angular/cli`

No terminal: `ng version`

# TypeScript



Extensão do javascript

Adiciona tipos

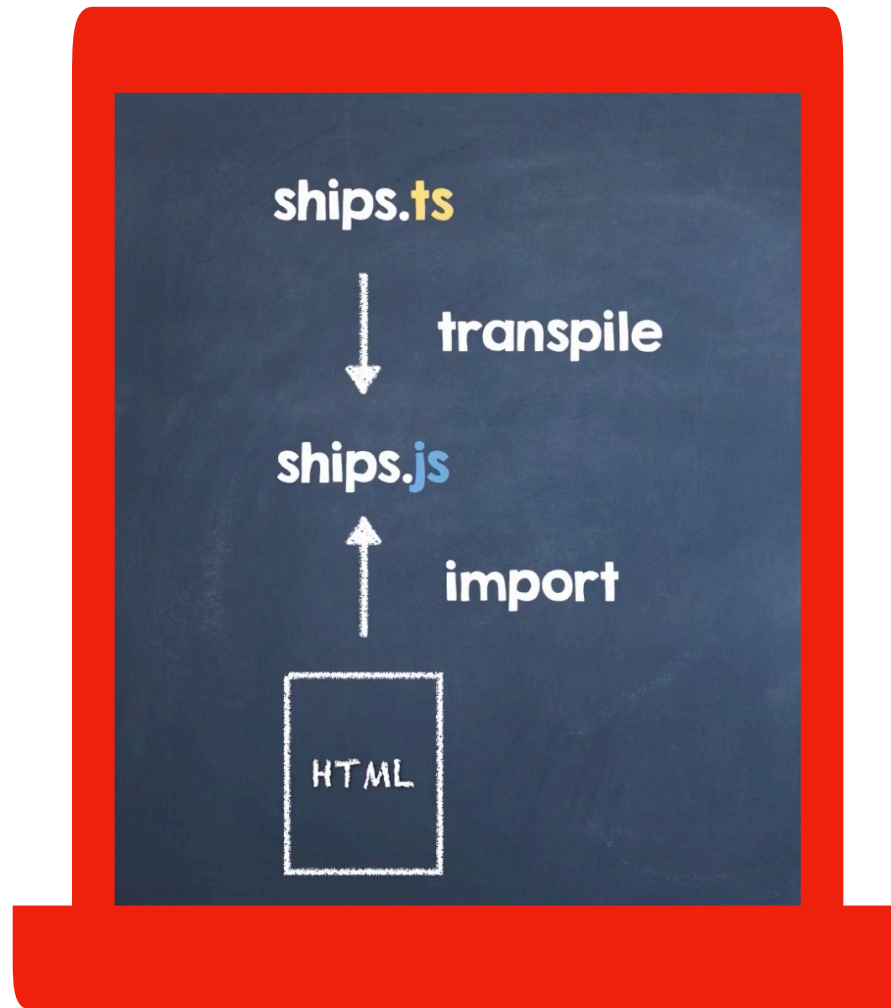
Classes e Interfaces

Javascript

```
let ano = 2021;  
incremento = "1";  
console.log(ano + incremento);
```

```
> 20211
```

# Estrutura básica



O navegador não entende Typescript

Para o que o código seja executado é necessário o processo de transpile, que converte o código em Javascript

# Declaração de variáveis



## Usando tipos

```
let nome: string;
```

```
nome = "joaquim";
```

```
nome = 10; (erro)
```

## Inferencia de tipos

```
let nome = "joaquim";
```

```
nome = 10; (erro)
```

Em javascript os erros só podem ser identificados em tempo de execução, já no typescript os erros são identificados em tempo de implementação.

## Tipos permitidos em Typescript

boolean

number

string

# Arrays



**Arrays podem ser declarados de três formas**

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

Indicando o tipo  
seguido de colchetes

```
let numeros: Array<number> = [1, 2, 3];
```

Usando a classe Array  
indicando o tipo

```
let numeros = [1, 2, 3];
```

Pela inferencia de tipos



# Mais sobre o ambiente de desenvolvimento

## VSCode



```
fetch.ts — danger-js

1  import * as debug from "debug"
2  import * as node_fetch from "node-fetch"
3
4  const d = debug("danger:networking")
5  declare const global: any
6  /**
7   * Adds logging to every fetch request if a global var for `verbose` is set to true
8   *
9   * @param {(string | fetch.Request)} url the request
10  * @param {fetch.RequestInit} [init] the usual options
11  * @returns {Promise<fetch.Response>} network-y promise
12  */
13  export function api(url: string | node_fetch.Request, init: node_fetch.RequestInit): Promise<node_fetch.Response> {
14    if (global.verbose && global.verbose === true) {
15      const output = ["curl", "-i"]
16
17      if (init.method) {
18        output.push(`-X ${init.method}`)
19      }
20
21      const showToken = process.env["DANGER_VERBOSE_SHOW_TOKEN"]
22      const token = process.env["DANGER_GITHUB_API_TOKEN"]
23
24      if (init.headers) {
25        for (const prop in init.headers) {
26          if (init.headers.hasOwnProperty(prop)) {
27            // Don't show the token for normal verbose usage
28            if (init.headers[prop].includes(token) && !showToken) {
29              output.push("-H", `${prop}: [API TOKEN]`)
30              continue
31            }
32            output.push("-H", `${prop}: ${init.headers[prop]}`)
33          }
34        }
35      }
36
37      if (init.method === "POST") {
38        // const body:string = init.body
```

# Primeiros passos



A primeira coisa a fazer é instalar o compilador do typescript. Isso é feito através do gerenciador de pacotes do node, o npm.

**npm install typescript -g**

No terminal, **tsc -v** pode ser usado para verificar se o compilador já está instalado e qual a sua versão.

**tsconfig.json**

Arquivo que pode conter as configurações básicas do compilador. Se ele for deixado vazio o compilador vai usar as configurações padrão.

**app.ts**

Os arquivos em typescript devem ter a extensão .ts

# Compilando os scripts



Para compilar um arquivo

```
tsc <nome do arquivo>
```

Para que o compilador fique monitorando arquivos e mudanças

```
tsc -w
```

# Um pouco mais sobre o tsconfig



**É possível fazer diversas configurações nesse arquivo, uma delas é indicar onde os arquivos .js devem ser gerados, organizando melhor o projeto.**

```
{
  "compilerOptions": {
    "outDir": "dist"
  }
}
```

# Exemplo



```
let nome: string;  
nome = "Antonio";  
  
console.log(nome);
```

**Para executar um script**

**node <nome do arquivo>;**

```
node dist/app
```

# Funções



Em typescript as funções, assim como no javascript, podem ter parâmetros e retornos. Além das funções normais, o typescript também implementa uma outra forma de escrever funções, as Arrow functions.

As funções podem ser nomeadas ou anônimas.

```
function qualNome(nome: string): void{  
    console.log("O nome é : " + nome);  
}
```

```
let numeroMaior = function(valor: number): boolean{  
    return numero > 10;  
}
```

```
console.log(numeroMaior(8));  
> true
```

```
console.log(numeroMaior("12"));  
> erro
```

# Funções



No typescript todos os parâmetros de uma função são obrigatórios, mas é possível marcar um parâmetro como opcional através do uso do operador ?

```
function exibir(nome: string, idade?: number): void{  
    let i = idade || 0;  
    console.log("O nome é : " + nome + ", a idade é : " + i);  
}
```

```
function exibir(nome: string, idade: number = 0): void{  
    let i = idade || 0;  
    console.log("O nome é : " + nome + ", a idade é : " + i);  
}
```

# Arrow functions



É uma forma abreviada de escrever as funções. As arrow functions não possuem a palavra chave ***function*** e podem até mesmo não ter a palavra chave ***return***.

Sintaxe básica:

```
(param1, param2, ..., paramn) => {comandos}
```

```
const soma = (numero1, numero2) => {  
  return numero1 + numero2;  
}
```

```
console.log(soma(5, 4));
```