

Téc em Desenvolvimento
de Sistemas Bilíngue

Desenvolver Código

Orientado a Objetos

UC4 | Prof. Vitor Hugo Lopes

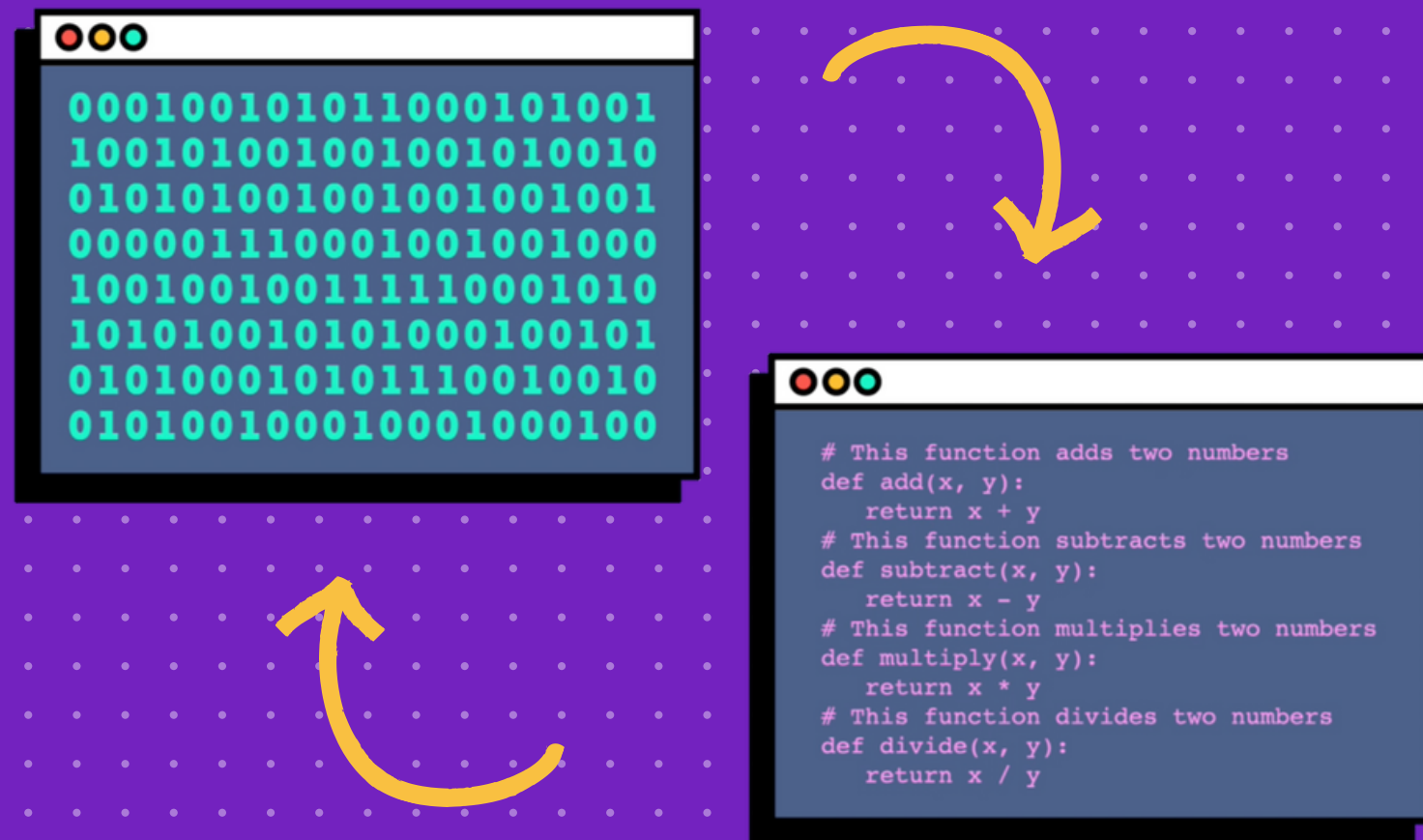
Linguagens de Programação



O que são?

- Conjuntos de sintaxes
- Permitem a comunicação entre humanos e máquinas
- Tal qual idiomas, cada linguagem **possuem variações** conforme o tempo e precisam de algum tipo de tradução para que sejam compreendidas corretamente

Linguagens de Programação



Como classificar?

- Conforme se afasta da linguagem de máquina e se aproxima da humana seu **nível de abstração** é maior.
- Podem ser classificadas em baixo nível e alto nível.

Linguagens de Programação



Como a máquina entende?

- **Compilador:** utilizado para converter o código em um EXECUTÁVEL (ex: C, Java)
- **Interpretador:** Utilizado quando o código é executado em tempo de execução (RUNTIME) (ex: Python, JS, PHP)
- **Transpilador:** Quando uma linguagem é convertida em outra (ex: Typescript)

Linguagens de Programação

O que são?

Conjunto de sintaxes utilizados para criar comandos na **comunicação** humano-máquina.

Como classificar?

São classificadas de **baixo** a **alto** nível sendo que, quanto mais alto, mais próximo à linguagem humana

Como compreender?

Existem 3 tipos:
Compiladores,
Transpiladores e
Interpretadores

Paradigmas de Programação



O que é?

- Paradigmas de programação são os conjuntos de características de uma linguagem que definem a forma de se pensar para a resolução de problemas.
 - Funcional
 - Declarativo
 - Orientado a Eventos
 - Orientado a Objetos

Paradigmas de Programação



Funcional

- É a ideia de se utilizarem **funções** para executar o código
- Neste paradigma, normalmente, todo o código é **dividido em funções** e o resultado final é a chamada de todas essas funções
- Exemplo: Clojure, Elixir, JS, TS, C, etc

Paradigmas de Programação



Declarativo

- Neste paradigma, espera-se que **se descreva** o que o código faz, mas não como ele faz
- O melhor exemplo deste paradigma são as linguagens de marcação: HTML, XML, etc.

Paradigmas de Programação



Orientado a Eventos

- Aqui o código depende que algo aconteça para ser executado
 - Você só acorda quando o despertador toca
 - Um formulário só envia as informações quando um usuário clica num botão
- Exemplos: C#, Java, JS Vanilla

Paradigmas de Programação

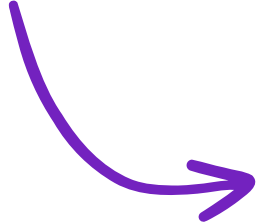


Orientado a Objetos

- Neste paradigma, o objetivo é tentar criar um sistema com modelos que mais se aproximem do mundo real
- Para isto, fazemos o uso de objetos e classes
- Exemplos: TS, Java, C++

JavaScript

A large, bold, black 'JS' logo is centered on a bright yellow rectangular background. The background is set against a purple background with a white dot pattern.

- Linguagem **interpretada** e **funcional**, inicialmente criada apenas para executar scripts em navegadores
- Por meio de **node.js** passa a ser possível ser utilizado fora de navegadores
- Possui tipagem dinâmica, ou seja, utiliza a **inferência de tipos**


valores são definidos de acordo com o valor recebido
- Devido à expansão do direcionamento, a linguagem possui algumas "**peculiaridades**".

JavaScript

JS

```
0.1 + 0.2
```

```
0.1 + 0.2 === 0.3
```

```
x = 1.0000000000000000001
```

```
x === 1
```

JavaScript

JS

```
typeof NaN  
NaN !== NaN
```

```
[] + []
```

```
[] + {}
```

```
{} + []
```

```
{} + {}
```

JavaScript

JS

```
Array(16)  
Array(16).join("quack")  
Array(16).join("quack" + 1)  
Array(16).join("quack" - 1) + " Batman"
```

TypeScript



TS

- Linguagem **orientada a objetos** de tipagem forte
- **Transpilada** para JavaScript
- Um verificador de **tipos estáticos** (static type checker) para JS
- Possui todas as funcionalidades do JS a disposição

TypeScript



TS

- Linguagem **orientada a objetos** de tipagem forte
- **Transpilada** para JavaScript
- Um verificador de **tipos estáticos** (static type checker) para JS
- Possui todas as funcionalidades do JS a disposição

Instalação do TS

Para instalar o TypeScript no seu computador podemos usar o comando

```
npm install -g typescript
```

Além disso todos os nossos arquivos devem terminar com a extensão **.ts**

Rodando o projeto

Para rodar o projeto é necessário primeiro transpilar o código para JavaScript:

```
tsc arquivo.ts
```

Depois o projeto deve ser rodado como qualquer outro código em JavaScript:

```
node arquivo.js
```

Que mão, né?

Podemos automatizar o processo de transpilar e executar o código. Para isso vamos precisar de um arquivo package.json.

```
npm init
```

Depois de responder - ou não - todas as perguntas do init, vamos criar um script dentro do arquivo para dar o start:

```
start: "tsc index.ts && node index.js"
```

Declarando variáveis comuns

No TS, usamos os mesmos declaradores do Javascript: **const** e **let**

Agora, precisamos colocar o **tipo** delas logo após o nome

```
//STRING
let name: string = "Vit"

//BOOLEAN
let isOk: boolean = true

//NUMBER
let age: number = 28
```

Arrays & Objetos

Para **arrays**, temos duas opções.

Para **objetos**, o tipo é bem parecido com a declaração. Nas **propriedades** do objeto, colocamos os seus respectivos tipos.

```
//ARRAY
let arr: Array<number> = [1,2,3]
let array: number[] = [1,2,3]

//OBJETO
let person: {name: string, age: number} =
{
    name: "Vitor",
    age: 28
}
```



Obrigado!

Dúvidas? Escreva para vhlopes@senacrs.com.br