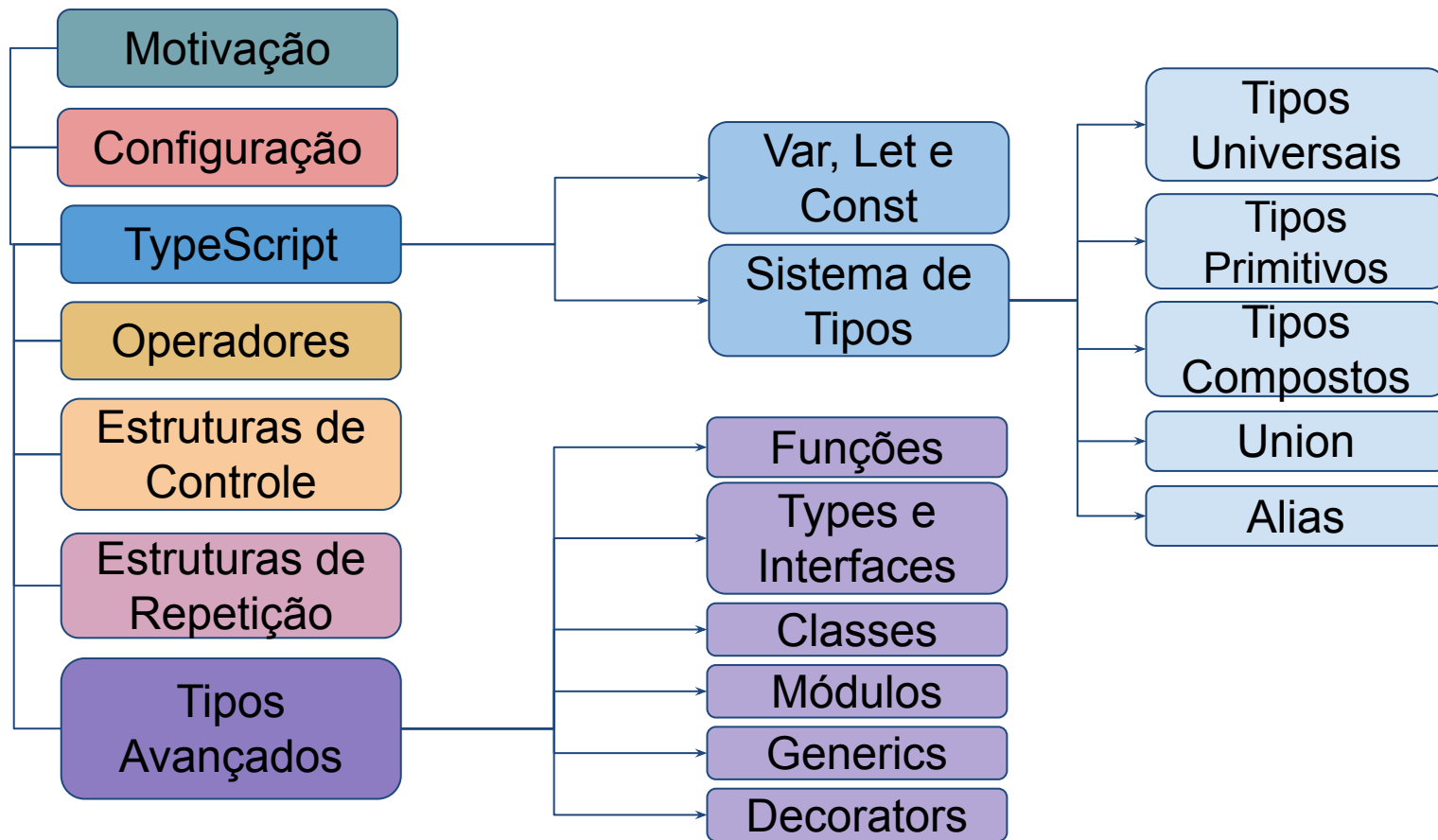


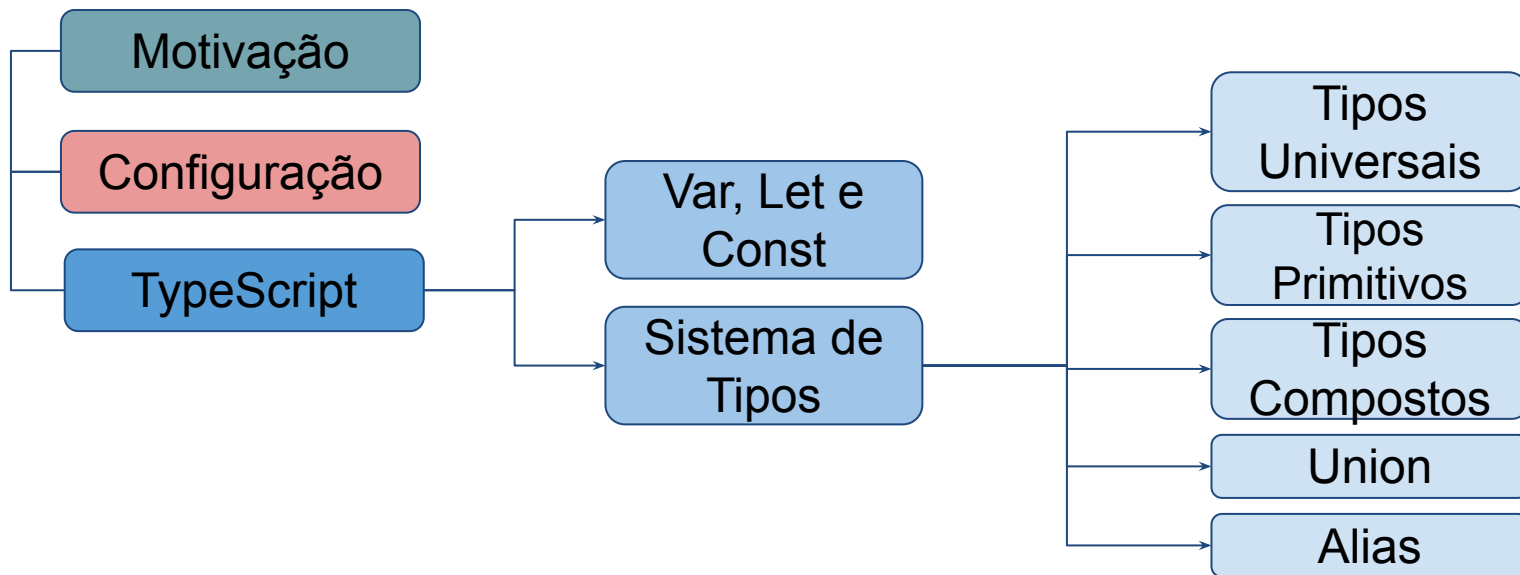


Maiara de Souza Coelho
Instituto de Computação
UFAM

Nossa Jornada



Tópicos Aula 1



JavaScript

- Quais os problemas do código?
 - <https://playcode.io/javascript>
 - <https://www.typescriptlang.org/>

```
function digaMeuNome (nome) {  
  console.log (nome) ;  
}  
  
digaMeuNome ("Beyonce", "Desconhecido") ;
```

JavaScript

- Quais os problemas do código?
 - <https://playcode.io/javascript>
 - <https://www.typescriptlang.org/>

```
function digaMeuNome (nome) {  
  console.log (nome) ;  
  console.log (nome.toUpperCase () ) ;  
}  
  
digaMeuNome (123) ;
```

Javascript X Typescript

- JavaScript:
 - Não identificou o tipo correto da variável
 - Erro de Execução no JavaScript. E se estiver na produção??
- TypeScript:
 - Identifica o erro em tempo de compilação

TypeScript

- Lançado pela Microsoft em 2012 e *OpenSource*.
- Linguagem de programação que engloba (*Superset*) o Javascript e adiciona novas especificações de sintaxe para definição e uso de tipos de dados
- Pré-processador de códigos JavaScript.
- Principal vantagem:
 - Ajuda no momento de depuração do código, prevenindo bugs ainda em tempo de desenvolvimento/compilação
- O TypeScript precisa ser transpilado para uma das versões em ECMAScript, pois não é reconhecido pelo navegador.



Vantagens do Typescript

- Feedback mais rápido de erros: verificador de tipos: analisa estruturas
- Liberdade com restrições, documentação precisa e ferramentas de desenvolvedor mais robustas
- Autocomplete da linguagem, muito boa no vscode: serviço de Linguagem
- Poder adotar gradualmente typescript em uma base de código: migração de código segura e tranquila
- Orientada à Objetos de verdade.

Instalações

- Visual Studio Code integração perfeita com o Typescript
- Node js e npm:
 - `node -v && npm -v`
- Compilador Typescript 5.0.2:
 - `npm i -g typescript@5.0.2 --save-dev`
 - `tsc -v`

Primeiro Arquivo TS

- criar um diretório cursoTS: `mkdir cursoTS`
- criar arquivo e abrir no VSCode e editar
- Compilar: `tsc aula1.ts`
- Executar: `node aula1.js`

```
const tituloCurso: string = "Fundamentos do Typescript";  
console.log("Bem vindo " + tituloCurso);
```

Configurando o Compilador TS

- O compilador do TypeScript é altamente configurável. Através do arquivo `tsconfig` é possível definir onde os arquivos `ts` e `js` ficam, bem como a versão do ECMAScript a ser utilizada, entre outras configurações.
- Gerar o arquivo `tsconfig.json`: `tsc --init` (pode ser criado manualmente)
- Criar diretórios `src` e `build` (ou `dist`) e configurar no `tsconfig`, respectivamente:
 - `rootDir = ./src`
 - `outDir = ./build` (ou `dist`)
- Os arquivos Typescript serão transpilados em javascript e estarão disponíveis em `build` (ou `dist`)
 - `tsc -w`

Configurando o Compilador TS

- Arquivo tsconfig.json criado manualmente

```
{  
  "compilerOptions": {  
    "outDir": "./build",  
    "target": "ES6",  
    "noEmitOnError": true,  
    "module": "commonjs",  
    "rootDir": "./src",  
  },  
}
```

Características Importantes

- Ponto e vírgula é opcional - indico o uso.
- Comentários:
 - `//` - uma linha
 - `/* */` - várias linhas
- Identificadores:
 - Não podem iniciar com dígitos mas, podem conter
 - Apenas os símbolos: `'_'` e `'$'`
 - Não podem ser palavras-chave da linguagem: `private`, `for`, `finally`, etc..
- Variáveis - `var`, `let` e `const`:
 - `var/let/const <nome_variavel>: Tipo = valor;`

Exemplos de Variáveis

Testar em: <https://www.typescriptlang.org/>

- `const UMACONSTANTE: number = 2;`
- `let UmaVariavel;`
- `var UmaOutraVariavel = "Alunos";`
- `let SouUmaVariávelComAcentuação = "acento";`
- `var 0_0 = "valido";`
- Funções como valores:

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

Var, let e const: Diferenças

- Testar em: <https://www.typescriptlang.org/>
- var

Hoisting - conceito do JavaScript para elevação da variável ao topo do escopo. Var tem escopo por função

```
var foraDoIf = 'Fora do if';  
if (true) {  
    var dentroDoIf = 'Dentro do if';  
    console.log(dentroDoIf);  
}  
console.log(foraDoIf);  
console.log(dentroDoIf);
```

Var, let e const: Diferenças

- Testar em: <https://www.typescriptlang.org/>
- let

```
let foraDoIf = 'Fora do if';  
if (true) {  
    let dentroDoIf = 'Dentro do if';  
    console.log(dentroDoIf);  
}  
console.log(foraDoIf);  
console.log(dentroDoIf);  
//ReferenceError: dentroDolf is not defined
```

Let - escopo por bloco
Variável limitada ao escopo do if.

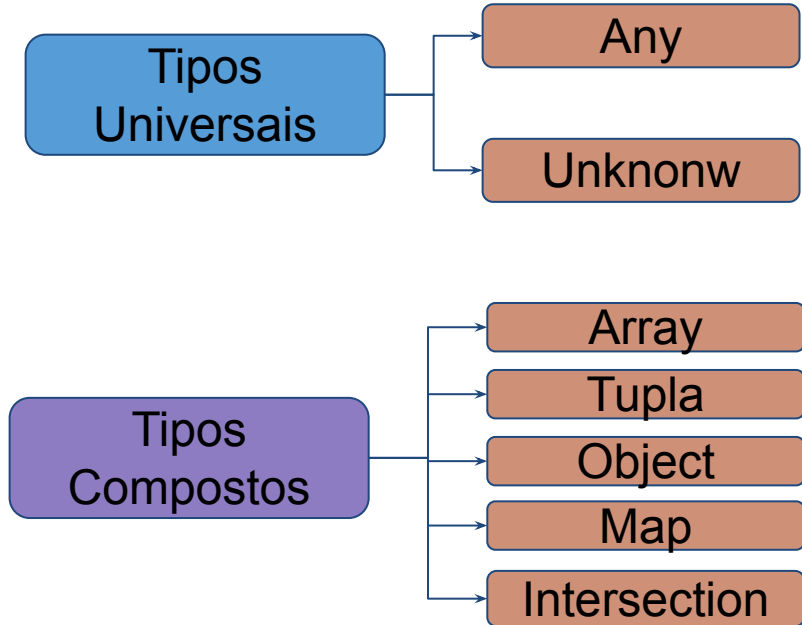
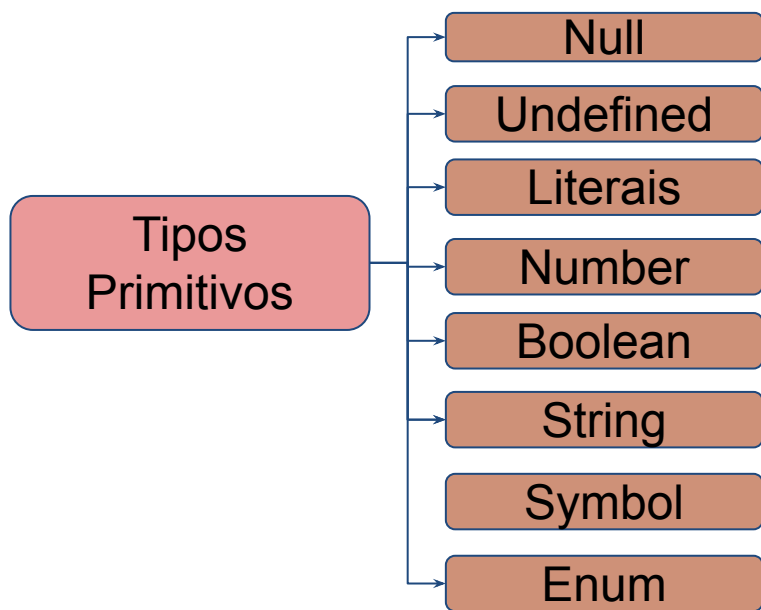
Var, let e const: Diferenças

- Testar em: <https://www.typescriptlang.org/>
- const

Imutável

```
const mensagem = 'MSG 1'
console.log(mensagem); // MSG 1
console.log(mensagem);
mensagem = 'MSG 2';
// TypeError: Assignment to constant variable.
```

Tipos



Tipos Universais: Any

- Any: Muito usado em integração com APIs de terceiros. Pode receber qualquer tipo de variável.
 - Solicita explicitamente ao TypeScript para não executar o verificador de tipos. Não se pode utilizar funções específicas de tipo, pois dão erro em tempo de execução, como no JavaScript.

Tipos Universais: Any

```
let variavelAny: any = "Variável"  
variavelAny = 34;  
console.log(variavelAny)  
variavelAny = true;  
console.log(variavelAny)
```

```
function cadastrarUsuario(nome:any): void{  
  console.log(`Usuário ${nome.toUpperCase()}  
  cadastrado`)  
}  
  
cadastrarUsuario("Maiara");
```

Tipos Universais: Unknown

- Unknown: Aceita qualquer tipo mas, faz verificação para propriedades específicas com uso de modificadores de tipo como: asserções de tipo ou uso de typeof.

```
function cadastrarUsuario(nome: unknown): void {  
  console.log(`Usuário ${nome.toUpperCase()}  
  cadastrado`)  
}  
  
cadastrarUsuario({nome: "Maiara"});
```

nome.toUpperCase is not
a function

Tipos Universais: Unknown

```
function cadastrarUsuario(nome: unknown): void {  
  if (typeof nome === "string") {  
    console.log(`Usuário ${nome.toUpperCase()}  
cadastrado`)  
  } else {  
    console.log("Não Cadastrado")  
  }  
}  
  
cadastrarUsuario("Maiara");  
cadastrarUsuario({});
```

Tipos Primitivos: Null, Undefined e Literais

- null: Faz referência a o primeiro e único valor do tipo Null.
 - `let nullV: null = null;`
- undefined: Denota o valor dado a todas variáveis não inicializadas.
 - `let undVar: undefined = undefined;`
- literais: São tipos de valores exatos, pode ser number, string, mas depois de declarado não pode mudar de valor.
 - `const literalV = "Literal";`

Tipos Primitivos: Number

- number: Representa valores de ponto flutuante IEEE 754 de precisão dupla de 64 bits.
 - `let numberV: number = 100;`
 - `let numberV = 100;`

```
let valor1: number = 10;
valor1 = 20.90;
let valor2: number;
valor2 = 15
console.log(valor1 + valor2);
console.log(valor1.toPrecision(2));
```


Tipos Primitivos: Boolean

- boolean: Valores true e false.
 - `let booleanV: boolean = true;`
 - `let booleanV = true;`

Tudo que for diferente de zero, string vazia, undefined, será verdadeiro.

```
let ativo : boolean
ativo = false;
ativo = true;
console.log(ativo)
```

```
let estaAutenticado: boolean = true;
estaAutenticado = false
let codeStatus: string = '';
estaAutenticado =
Boolean(codeStatus);
console.log(estaAutenticado);
```

Tipos Primitivos: String

- string: Sequência de caracteres armazenados como unidades de código Unicode UTF-16
 - `let stringV: string = "Variável do tipo string";`
 - `let stringV = `Meu nome é: ${name}`;`

```
let tecnologia: string;  
tecnologia = "JavaScript"  
console.log(tecnologia.length)  
console.log(tecnologia.indexOf("J"))  
console.log(tecnologia.indexOf("d"))  
console.log(tecnologia.toLowerCase())
```

Tipos Primitivos: Symbol

- symbol: Representa tokens exclusivos que podem ser usados como chaves para propriedades de objetos.
 - `let symbolV: symbol = Symbol("key");`
 - `let symbolV = Symbol("key");`

Symbols
são únicos

```
let sym1 = Symbol("key");  
let sym2 = Symbol("key");  
sym1 === sym2;
```

```
const sym3 = Symbol();  
let obj3 = {  
  [sym3]: "value",  
};  
console.log(obj3[sym3]);
```

Tipos Primitivos: Enum

- enum: São subtipos definidos pelo usuário.
 - Numérico, String e Heterogêneo

```
enum cores {  
    branco = "#FFFFFF",  
    preto = "#000000"  
}  
  
console.log(cores["branco"])
```

```
enum permissao{  
    ADMIN,  
    USUARIO,  
    SUPORTE  
}  
  
console.log(permissao.ADMIN)  
console.log(permissao.SUPORTE)
```

Tipos Primitivos: Enum

```
enum Heterogeneous {  
  Segunda = 'Segunda-feira',  
  Terca = 1,  
  Quarta,  
  Quinta,  
  Sexta,  
  Sabado,  
  Domingo = 18,  
}  
  
console.log(Heterogeneous["Segunda"]);  
console.log(Heterogeneous.Segunda);
```

Tipagem Dinâmica

```
let userId;  
userId = 123;  
console.log(userId)  
userId = "Aluno";  
console.log(userId)
```

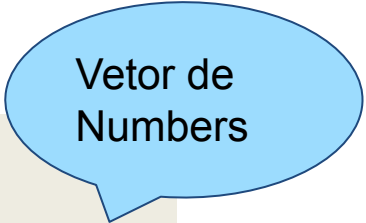
let userId:number

let userId:string

Tipos Compostos: Arrays

- Array: Lista de elementos do mesmo tipo.
 - `let frutas:string[] = ["maçã", "laranja"];`
 - `let frutas:Array<string> = ["maçã", "laranja"];`
 - `let frutas = ["maçã", "laranja"];`

```
let numeros: number[];  
numeros = [1, 5, 21, 30]  
console.log(numeros);  
numeros.push(55);  
console.log('Meus numeros: ', numeros)  
console.log(numeros[0])
```



Vetor de
Numbers

Tipos Compostos: Arrays

- Union: number|boolean|null|undefined

```
const elementos = [true, null, undefined, 42];  
elementos.push(["Nome"]);  
console.log('Meus numeros: ', elementos)
```


Tipos Compostos: Arrays Multidimensionais

```
let arrayMulti: number[][];  
arrayMulti = [  
    [1, 2, 3],  
    [4, 5, 6],  
    [7, 8, 9]  
];  
console.log(arrayMulti)
```

Tipos Compostos: Tuplas

- Tupla: Tuplas são listas fixas de elementos heterogêneos.
 - `let par: [string, number] = ["par1", 123];`
 - `let par = ["par1", 123];`
 - `let list: [nome: string, idade: number, email: string]
= ['Bill Gates', 65, 'bill@teste.com'];`
 - Acesso, inserção e remoção como em arrays

Tipos Compostos: Objeto

- Objeto: É qualquer coisa que não seja um tipo primitivo, (pode ser anônimo).

```
let usuario: object = {  
  nome: "Matheus Silva",  
  email: "teste@teste.com",  
  idade: 25  
}  
console.log(usuario.nome);
```

```
let usuario = {  
  nome: "Matheus Silva",  
  email: "teste@teste.com",  
  idade: 25  
}  
console.log(usuario.nome);
```

Tipos Compostos: Mapeado

- Mapeado: Derivar um tipo de objeto em outro.
 - `let v: map = { a: "a", b: "b", c: "c" };`

```
type x = 'a' | 'b' | 'c';  
type map = { [keys in x]: string };  
let v: map = { a: "a", b: "b", c: "c"};  
console.log(v.a);
```

Tipos Compostos: Interseção

- Interseção: Conjunto dos valores em comum.
 - `let intersecao: tipo1 & tipo2;`

```
type Info = {  
  id: number;  
  nome: string;  
  descricao?: string;  
}
```

```
type Categoria = {  
  slug: string;  
  quantidadeProduto:  
  number;  
}
```

Tipos Compostos: Interseção

```
type ProdutoInfo = Info & Categoria;  
const novoProduto: ProdutoInfo = {  
  id: 54321,  
  nome: "Teclado RGB",  
  slug: "teclado-mecanico",  
  quantidadeProduto: 10,  
}  
console.log(novoProduto);
```

União (Union)

- União: Permite adotar mais tipos para uma variável.
 - `let` `userId`: `number` | `string` = `123`;

```
let exemploVariavel: ( string | number );  
exemploVariavel = 123;  
console.log(exemploVariavel);  
exemploVariavel = "ABC";  
console .log(exemploVariavel);
```

União (Union)

```
function acessarUsuario(usuario: string|string[]) {  
  if (typeof(usuario) == "string") {  
    console.log(usuario, "acessado" );  
  } else {  
    for (let i = 0; i < usuario.length; i++) {  
      console.log(usuario[i], "acessado" );  
    }  
  }  
}  
  
acessarUsuario(["user1", "user2"]);  
acessarUsuario("user1");
```


Apelido (Alias)

- Possibilidade de adotar nomes mais fáceis para tipos unions reutilizados.

```
let data1: boolean | number | string | null | undefined;  
let data2: boolean | number | string | null | undefined;  
let data3: boolean | number | string | null | undefined;
```



```
type Data: boolean | number | string | null | undefined;  
let data1: Data;  
let data2: Data;  
let data3: Data;
```

Apelido (Alias)

- Alias podem ser combinados

```
type Id = number | undefined | null;  
type IdTalvez = Id | number | string;
```

Exercício 1

- Crie um array que represente livros de uma biblioteca.
- Imprima seu nome, preço e categoria de todos os livros.

Exercício 2

- Defina a categoria como um enum e como opcional.

Exercício 3

- Nomes de livros com apenas uma palavra devem estar em maiúsculo.
- Caso contrário em minúsculo.
- Use o operador ternário condicional

Exercício 4

- Crie um map de um tipo união entre os tipos autor e livro.
- Acesse cada biblioteca e imprima os livros de cada uma.

Por hoje é só, pessoal!

Obrigada

Dúvidas: Slack

maiara@icomp.ufam.edu.br

github: mayara-msc@hotmail.com