TYPESCRIPT

O Typescript é um superset do javascript, ou seja, um super conjunto desenvolvido pela Microsoft, para auxiliar no desenvolvimento de código Javascript, tornando tudo mais fácil, com menos linhas e mais estabilidade, deixando o código menos suscetível a erros.

Uma das grandes vantagens do typescript é o foco nas tipagens, embora não seja obrigátorio, é importante o desenvolvedor utilizar tipagem forte, assim como nas linguagens C# e Java, por exemplo, pois as tipagens auxiliam na transpilação do código typescript que este é transformado em javascript. Desta forma o código será validado e avisado caso haja erros previavente configurados pelo desenvolvedor.

Por ser *Open Source* o typescript tem seu código em domínio público, acessível a qualquer desenvolvedor que tenha o interesse de criar extensões e aceleradores.

Como o código typescript é transformado em javascript, ele acaba sendo suportado por todas as plataformas que suportam javascript.

1. Tipos básicos

boolean, string, number; coleções: number[], Array<number>, any[], string[], Array<string>;

2. Declaração de variáveis e constantes

O que conhecemos como 'var' para declaração de variáveis, tornou-se 'let' no typescript, ficando da seguinte forma:

```
let variavel: number;
variavel = 1;

let nome: string = 'Avanade';
```

E constantes, que podem ser criadas nos métodos ou no arquivo, não podendo ser utilizada diretamente na classe. As contantes podem ser utilizadas no nosso modulo ou podem ser utilizadas por outros módulos, basta esta ser importada no modulo que será utilizada, porém para isto deve existir o prefixo 'export' para que seja permitida a importação deste modulo em outros módulos. Para utilização segue a sintaxe:

```
const VALOR: number = 10; //constante privada
export const NOME: string = 'Avanade'; // constante pública
export const NUMEROS: number[] = [1,2,3]; // constante pública
```

3. Criação de funções e funções genéricas

```
function funcaoSemParametro(): string {
    return 'string';
}

function funcaoComParametro(num1: number, num2: number) {
    let total: number;
    total = num1 + num2;
```

```
}
function funcaoComParamRetorno(num1: number, num2: number):
number {
    return num1 + num2;
}
```

Na chamada da função genérica o seu tipo é informado ao "T", pois como o próprio nome diz se trata de uma função genérica que não conhece o tipo de informação que irá trabalhar.

```
function funcaoGenerica<T>(param: T): T{
    return param;
}
[…]
let retornoFuncao: string = funcaoGenerica<string>("Ola
Avanade");
```

4. **Criação de interfaces**

```
interface INova{
     metodoUm(param: number): number;
}
```

Vale lembrar que se não usarmos o "export" a nossa interface será acessível apenas ao arquivo ts onde ela foi criada, portanto a interface só poderá ser importada e implementada, em uma classe de outro arquivo .ts, caso ela possua o modificador "export" para torna-la disponível em outro modulo.

5. **Criação de classes**

O export é o modificador da nossa classe, para torna-la pública.

```
export class PrimeiraClasse{
     //propriedades da classe
     Id: number;
     Nome: string;
     Sobrenome: string;
     constructor(private nome: string,
                  private sobrenome: string) {
          this.Nome = nome;
          this. Sobrenome = sobrenome;
     }
     //método sem retorno
     primeiroMetodo(): void{
     }
     //método retornando string
     fullName(): string{
          return this.Nome + ' ' + this.Sobrenome;
     }
     //método recebendo parâmetros
     fullParamName(nome: string, sobrenome: string): string {
     return nome + ' ' + sobrenome;
     }
}
```