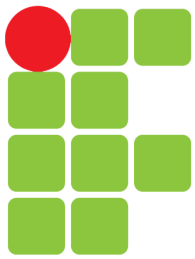

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PIAUÍ

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
PERÍODO III - PROGRAMAÇÃO PARA INTERNET
PROF.: ELY MIRANDA
LUCAS GOMES DE OLIVEIRA

Exercícios

1) Verifique nas alternativas abaixo quais compilam ou não. Explique o motivo:
a) Não compila Uma vez atribuído um valor numérico a uma variável, o compilador não irá permitir que esta mesma variável seja atribuída com um valor do tipo string.

```
let a = 10;
a = "2";
```

b) Compila

```
let b: any = 10;
b = 2;
```

c) Compila

```
let c: number = 10;
c = 2;
```

---

2) Dada a função `soma` abaixo, tente executar os *scripts* das alternativas e exiba os eventuais resultados:

```
function soma(x: number, y?: any): number {
  return x + y;
}
```

a) 3

```
console.log(soma(1, 2));
```

b) “12” (tipo *string*)

```
console.log(soma(1, "2"));
```

c) NaN (Not a Number)

```
console.log(soma(1));
```

---

3) Crie uma `enum` com as siglas dos estados “PI”, “CE”, “MA” e implemente as duas alternativas abaixo:

a) Crie um laço usando for para imprimir esses valores

```
enum Estados {
    "PI",
    "CE",
    "MA"
};

for (let i: number = 0; i < Object.keys(Estados).length / 2; i ++) {
    console.log(Estados[i]);
}
```

b) Crie um laço que imprima os índices dessa enum e diga o que aconteceu

```
enum Estados {
    "PI",
    "CE",
    "MA"
};

for (let i: number = 0; i < Object.keys(Estados).length / 2; i ++) {
    let estado: string = Estados[i];

    console.log(Object.keys(Estados).indexOf(estado));
}
```

---

4) Sobre enums, implemente o seguinte:

a) Crie uma `enum` chamada `DiasSemana` com os valores representando os dias da semana segunda a domingo

b) Crie um `namespace` com mesmo nome e dentro dele crie uma função chamada `isDiaUtil` recebe um parâmetro do tipo `DiasSema` e retorna `false` se for um sábado ou domingo e retorna `true` caso contrário

c) Escreva também um *script* que declara uma variável do tipo da `enum` e que testa a função `DiasSemana.isDiaUtil()`.

```
enum DiasSemana  {
   "Segunda",
   "Terça",
   "Quarta",
   "Quinta",
   "Sexta",
   "Sábado",
   "Domingo"
};

namespace verificarDiasSemana {
   export function isDiaUtil(day: any) {
       if (day == "Sábado" || day == "Domingo") {
           return false;
       }
       return true;
   }
}

for (let i: number = 0; i < 7; i++ ) {
   console.log(`${DiasSemana[i]}:
${verificarDiasSemana.isDiaUtil(DiasSemana[i])}`);
}
```

---

5) Crie uma função chamada exibir receba como parâmetro um "*rest parameter*" representando strings. A função deve exibir no log cada um dos elementos do "*rest parameter*". Chame a função usando diferentes quantidade de parâmetros conforme abaixo:

exibir("a", "b");
exibir("a", "b", "c");
exibir("a", "b", "c", "d");

```
function exibir(...input: any): void {
    //console.log(input);

    for (let i of input) {
        console.log(i);
    }
}

exibir('a', 'b');
exibir('a', 'b', 'c');
exibir('a', 'b', 'c', 'd');
```

---

6) Converta em *arrow function* a seguinte função:

```
function ola() {
    console.log("Olá");
}
```

Função convertida:

```
let ola = (): any => {
    console.log("Olá");
}
```

---

7) Dado método `filter` dos arrays, crie uma implementação usando *arrow function* que filtre todos os elementos pares do *array* abaixo:

```
const array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];
```

Implementação da *arrow function*:

```
let array: Array<number> = [];

for (let i: number = 0; i < 15; i ++) {
    array.push(i + 1);
}

function isPair(value: any) {
    return value % 2 == 0;
}

let pair: any = (): void => {
    console.log(array.filter(isPair));
}

pair();
```

---

8) Crie uma classe chamada `MeuNumero` tenha:
a) Um atributo chamado `numero` que seja somente leitura e inicializado no construtor
b) Um método chamado `getInteiro` que retorna a parte inteira do atributo `numero`
c) Um método chamado `getDecimal` que retorne a parte decimal do atributo `numero`
Dica: utilize a função `Math.floor(n)`
d) Instancie uma classe `MeuNumero` e teste os métodos da classe.

```
class MeuNumero  {
   numero: number;
```

```
    constructor(numero: any) {
        this.numero = numero;
    }

    getInteiro() {
        return Math.floor(this.numero);
    }

    getDecimal() {
        return (this.numero - Math.floor(this.numero)).toFixed(2);
    }

}

const numero = new MeuNumero(1.2302903);
console.log(`Inteiro: ${numero.getInteiro()}`);
console.log(`Decimal: ${numero.getDecimal()}`);
```

---

9) Crie uma classe chamada `Transacao` que tenha:
a) Um atributo chamado `valor` e um outro chamado `desconto`, ambos somente leitura;
b) Um método que calcule e retorne o valor do desconto aplicado ao valor original:

```
valor * (1-desconto/100).
```

c) Crie métodos de acesso get para ambos os atributos.
d) Instancie uma classe `Transacao` e teste seus métodos

```
class Transacao {
    readonly valor: number;
    readonly desconto: number;

    constructor(valor: number, desconto: number) {
        this.valor = valor;
        this.desconto = desconto;
    }

    getValor(): void {
        console.log(`Valor => ${this.valor}`);
    }

    getDesconto(): void {
        console.log(`Desconto => ${this.desconto}`);
    }
```

```
    Desconto() {
        return this.valor * (1 - this.desconto / 100);
    }
}

let op = new Transacao(34, 89);
console.log(`Valor apos o desconto: ${op.Desconto().toFixed(2)}`);
```