INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA PIAUÍ	**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PIAUÍ** **Curso: ADS** **Disciplina: Programação Orientada a Objetos** **Professor: Ely**

Exercício 02

Para as questões abaixo, crie um novo projeto usando TypeScript (tsc --init) e um ou mais arquivos com extensão .ts

1. Qual a diferença entre tipagem dinâmica e tipagem estática?

   R: A tipagem dinâmica é aquela que verifica os tipos de dados apenas em tempo de execução, assim permitindo com que uma variável mude de tipo durante a execução do programa. A tipagem dinâmica é utilizada nas linguagens de Python e JavaScript, por exemplo. Já a tipagem estática é aquela que verifica os tipos de dados durante a compilação do código, ou seja, caso haja uma incompatibilidade de tipos, o compilador irá sinalizar um erro antes do programa ser executado. A tipagem estática é utilizada nas linguagens de Java e TypeScript, por exemplo.

2. Qual o principal problema do uso de tipagem dinâmica?

   R: O principal problema no uso dessa tipagem é que os erros de tipo só são descobertos em tempo de execução. O que pode levar a um erro a passar despercebido durante o desenvolvimento de um programa e só aparecer quando este já estiver em uso, assim levando ao aparecimento de bugs e falhas na produção.

3. Pesquise um exemplo na internet em que a tipagem dinâmica pode ser problemática.

   R: Um problema comum em linguagens de tipagem dinâmica, como o JavaScript, ocorre quando uma função que espera que um tipo de dado receba outro. Isso não causa um erro de compilação, mas em certos casos pode ocorrer de o código não falhar de imediato com um erro de tipo, mas produzir um resultado inválido. Considerando o exemplo do código de cálculo de raiz quadrada apresentado, tem-se que ele é capaz de converter uma string como parâmetro da função quando esta é um valor numérico e realizar a operação corretamente, no entanto caso o valor digitado seja um texto como “quatro” o resultado é um NaN que pode causar comportamentos inesperados caso se propagasse em um código mais amplo. Em uma linguagem de tipagem estática, como TypeScript, a primeira chamada com “4”

já iria gerar um erro de compilação, assim prevenindo o problema antes que ele pudesse ocorrer em tempo de execução.

Códigos usados para a resolução da questão se encontram nos arquivos externos (q3_ex1.js,q3_ex2.js,q3_ex3.ts).

4. Crie uma variável chamada idade do tipo number e tente atribuir a ela um valor string. O que acontece?

   R: Vai ocorrer um erro de tipagem durante a compilação. O que vai levar o compilador do TypeScript a impedir essa operação porque a variável idade é estaticamente tipada, e a atribuição viola essa regra.

   Código usado para a resolução da questão se encontra no arquivo externo (q4.ts).

5. Agora crie a variável nome sem declarar o tipo (apenas let nome = "Ely";). Qual o tipo inferido pelo TypeScript?

   R: O tipo inferido pelo TypeScript para a variável nome é string.

   Código usado para a resolução da questão se encontra no arquivo externo (q5.ts).

6. Pesquise e exemplifique porque dizemos que a linguagem C, mesmo tendo tipagem estática, possui tipagem fraca.

   R: A linguagem C é considerada de tipagem fraca pois permite a conversão implícita de tipos de dados sem a necessidade de um cast explícito. Por exemplo, é possível somar um int com um char,e o compilador de C fará a conversão do char para o seu valor numérico correspondente, através da codificação feita através da tabela ASCII, para realizar a soma, o que pode levar a comportamentos inesperados. Isso contrasta com linguagens de tipagem forte que exigiriam uma conversão explícita para evitar erros.

7. Poderíamos dizer que a tipagem do TypeScript é fraca por uma variável do tipo number aceitar tanto inteiros como ponto flutuante?

R: Não, a tipagem do TypeScript não é fraca por esse motivo. Visto que uma linguagem é considerada fraca quando ela permite a conversão implícita de tipos de dados diferentes, como string para number. No caso do TypeScript, o tipo number foi projetado para comportar tanto valores números inteiros como de ponto flutuante, portanto fazendo com que estes pertençam à mesma categoria de tipo e por isso não há conversão implícita quando um number aceita os dois.

8. Execute os exemplos abaixo em um ambiente de JavaScript puro:
    a. Exemplo 1:

```
let a = 10;
let b = "5";

console.log(a + b);
```

    b. Exemplo 2:

```
let x = true;
let y = 2;
```

```
console.log(x + y);
```

c. Exemplo 3:

```
console.log(0 == false); // true
console.log("" == false); // true
console.log(null == undefined); // true
```

Quais os resultados? Teça comentários a respeito.

R: No exemplo 1, o JavaScript realizou a coerção de tipos para tornar a operação possível. Nesse caso, ocorreu a conversão do número 10 para a string “10” que foi concatenado pelo operador de adição a string “5”, retornando a string “105” como resultado. Já no exemplo 2, a coerção que o JavaScript realizou para a operação fez com que o operador booleano true fosse convertido para o seu equivalente numérico, que é o 1, o qual foi somado com o número 2 para retornar o número 3 como resultado. Já no exemplo 3, o operador foi utilizado o operador “==” que realiza comparações de valores após a conversão de tipos e ignorando os tipos de dados envolvidos, através dessas comparações o resultado obtidos foram três booleanos “true” pois 0 é considerado igual a false assim como uma string vazia e null é considerado igual a undefined.

Códigos usados para a resolução da questão se encontram nos arquivos externos (q8_ex1.js,q8_ex2.js,q8_ex3.js).

9. Execute os dois mesmos exemplos em um ambiente TypeScript. Verifique o que acontece e comente sobre.

R: No exemplo 1, o código foi executado sem mostrar erro, não sei se isso é coisa do playground, creio que os tipos sejam compatíveis para conversão. Já no exemplo 2, ocorreu um erro de compilação, já que a verificação de tipo estática do TypeScript identifica que uma operação com tipos incompatíveis está sendo realizada. Já no exemplo 3, o erro de compilação ocorre pois nas as duas primeiras comparações são feitas entre tipos sem overlap. Esses últimos dois exemplos demonstram a principal vantagem do TypeScript: ele previne erros que só seriam detectados em tempo de execução no JavaScript, garantindo maior segurança e previsibilidade ao código.

Códigos usados para a resolução da questão se encontram nos arquivos externos (q9_ex1.js,q9_ex2.js,q9_ex3.js).

10. Declare variáveis chamadas nome, salário e linguagem. Inicialize-as com os valores “Ely”, 2.000 e TypeScript. Gere uma a saída abaixo, mantendo a quebra de linhas usando template strings:

Ely
My payment time is 2000.00 and
my preffered language is TypeScript

R: Resposta no arquivo externo (q10.ts).

11. Os tipos de dados em TypeScript podem ter métodos auxiliares. Faça o que se pede abaixo:
    a. Crie uma variável mensagem do tipo string e
        i. Inicialize-a com o valor "TypeScript É MUITO LEGAL!";
        ii. Exiba a mensagem toda em minúsculas. https://www.tutorialspoint.com/typescript/typescript_string_touppercase.htm
        iii. Exiba a quantidade de caracteres da string. https://www.tutorialspoint.com/typescript/typescript_string_length.htm

R: Resposta no arquivo externo (q11_a.ts).

    b. Crie uma variável do tipo number chamada pi como um número inicialize-a com o valor 3.1415. Exiba apenas 2 casas decimais. https://www.geeksforgeeks.org/typescript/typescript-tofixed-function/

R: Resposta no arquivo externo (q11_b.ts).

Não vamos esquecer de pensar em objetos. Nas próximas questões, relembre a declaração de classes e instanciação de objetos e faça o que se pede.

12. Crie uma classe Pessoa com:
    a. atributos nome e idade;
    b. um método apresentar() que retorna uma string como:
       Meu nome é Ely e tenho 46 anos.

    Instancie uma classe, atribua valores aos atributos e imprima o resultado do método apresentar().

    R: Resposta no arquivo externo (q12.ts).

13. Crie uma classe Produto com:
    a. Atributos nome e preco;
    b. Um método aplicarDesconto(percentual: number) que retorna o preço com desconto aplicado;
    c. Um método chamado emitirOrcamento() que chama o método acima e retorna uma string como:
       Produto: Camisa, Preço: R$ 100.00
       Desconto: 10% → Novo preço: R$ 90.00
    d. Desafio: pesquisar como exibir os números com decimais usando vírgula com o método toLocaleString e estilo moeda em real.

    Instancie uma classe, atribua valores aos atributos e imprima o resultado do método emitirOrcamento().

    R: Resposta no arquivo externo (q13.ts).

14. Estude como funciona um “if” em TypeScript e o operador %. Crie uma classe Numero com:
    a. Um atributo valor;
    b. Um método que ehPar() que retorna verdadeiro ou falso dependendo do valor do atributo;
    c. Um método ehImpar() que retorna verdadeiro ou falso dependendo do valor do atributo.

    Instancie objetos Numero, inicialize o atributo valor e exiba os resultados dos métodos.

    R: Resposta no arquivo externo (q14.ts).

Para as próximas questões, pesquise e configure o seu arquivo de configuração do TypeScript com as opções abaixo. Faça testes com as mudanças e perceba a diferença após a configuração.

15. Altere o local em que os arquivos *.js são gerados para a pasta build: opção outDir

    R: Resposta no arquivo externo (tsconfig.json).

16. Alterne a opção allowUnreachableCode com os valores true/false, teste o código abaixo e descreva o que acontece:

```
function {
   let x: number = 10;

   console.log("Início do programa");

   if (x > 5) {
      console.log("x é maior que 5");
      // Tudo abaixo desse return nunca será executado
      return;
      console.log("Essa linha é inatingível!");
   }

   console.log("Fim do programa");
}
```

    https://www.typescriptlang.org/pt/tsconfig#allowUnreachableCode

    R: Com o "allowUnreachableCode": false é mostrado um erro de compilação na linha console.log("Essa linha é inatingível!"), avisando que o código não pode ser executado. Já com “allowUnreachableCode": true tem-se o desaparecimento do erro de compilação e o código será transpilado normalmente.
    Código usado para auxiliar na resolução da questão se encontra no arquivo externo (q16.ts), já a resposta se encontra no arquivo externo (tsconfig.json).

17. Alterne a opção noImplicitAny com valor true/false, teste o código abaixo teste o código abaixo e descreva o que acontece:

```
let valor; // Tipo implícito "any"
valor = 10;
valor = "teste";
console.log(valor);

let outroValor: number; // Tipo
```

declarado outroValor = 20;

console.log(outroValor);

https://www.typescriptlang.org/pt/tsconfig#noImplicitAny

R: Com o "noImplicitAny": false não é mostrado erro, pois a variável valor recebe o tipo any implicitamente. Já com “noImplicitAny": true tem-se um erro de compilação na linha let valor;, avisando que a variável não pode ter o tipo any implicitamente.
Código usado para auxiliar na resolução da questão se encontra no arquivo externo (q17.ts), já a resposta se encontra no arquivo externo (tsconfig.json).

18. Alterne a opção strictNullChecks para true/false e descreva o que acontece com o exemplo abaixo:

let nome: string;

nome = "Ely";

console.log("Nome:", nome);

R: Com o "strictNullChecks": false O código irá funcionar. Já com “strictNullChecks": true o TypeScript irá gerar um erro na linha let nome: string;, pois a variável não foi inicializada.

Código usado para auxiliar na resolução da questão se encontra no arquivo externo (q18.ts), já a resposta se encontra no arquivo externo (tsconfig.json).

19. Altere o atributo target com o valor ES3. Além disso, utilize a classe de um dos exercícios anteriores e veja como ela é transpilada para JS;

    R: Usando o código da questão 12 como exemplo, tem-se que ele com o "target": “ES2016” o JavaScript gerado na saída será uma classe moderna (class Pessoa {...}). Já com “target": “ES3” o JavaScript gerado será muito mais longo, usando funções construtoras e protótipos (function Pessoa(...) {...}), pois o ES3 não suporta a sintaxe de classes.

    R: Resposta no arquivo externo (tsconfig.json) com o código do arquivo externo (q12.ts) sendo usado como exemplo.

20. Configure seu projeto para que seja possível realizar depuração alterando o atributo sourceMap.
    https://www.youtube.com/watch?v=4oQutHz96is

    Execute pequenas depurações em trechos de código

    R: Resposta no arquivo externo (tsconfig.jso