1) - Tipagem Estática: Na tipagem estática, os tipos das variáveis são verificados em tempo de compilação, ou seja, antes do programa ser executado. Nesse modelo, o programador precisa declarar explicitamente os tipos das variáveis no momento da criação, e o compilador verifica se as operações realizadas entre diferentes tipos são compatíveis.

- Tipagem Dinâmica: Na tipagem estática, os tipos das variáveis são verificados em tempo de compilação, ou seja, antes do programa ser executado. Nesse modelo, o programador precisa declarar explicitamente os tipos das variáveis no momento da criação, e o compilador verifica se as operações realizadas entre diferentes tipos são compatíveis.

2) Erros de tipo em tempo de execução: Uma das desvantagens mais significativas da tipagem dinâmica é que os erros de tipo geralmente só são detectados durante a execução do programa, quando as operações incompatíveis entre tipos são realizadas. Isso pode levar a falhas inesperadas e comportamentos imprevisíveis em tempo de execução.

3)

```
numero1 = input ("Digite o primeiro número: ")

numero2 = input ("Digite o segundo número: ")

soma = numero1 + numero2

print ("A soma é:", soma)
```

Neste exemplo, o programa solicita ao usuário que insira dois números. No entanto, a função input() retorna uma string, não um número. Como resultado, quando você tenta somar as variáveis numero1 e numero2, você não está realizando uma adição numérica, mas sim uma concatenação de strings.

Se o usuário digitar, por exemplo, "5" e "10", o resultado da soma será "510", em vez de 15.

Isso acontece porque a tipagem dinâmica do Python permite que as variáveis mudem de tipo conforme o contexto. Embora isso possa oferecer flexibilidade, também pode levar a resultados inesperados quando não há uma conversão explícita de tipos.

4) C é uma linguagem de programação tipada estaticamente, pois o tipo das variáveis é definido em tempo de compilação e as operações entre tipos incompatíveis resultarão em erros de compilação.

No entanto, o C também é considerado uma linguagem de tipagem fraca, pois permite conversões automáticas entre diferentes tipos de dados, muitas vezes sem exigir uma conversão explícita. Isso pode levar a resultados inesperados e problemas se o programador não estiver ciente das regras de conversão.

Por exemplo, no C, você pode multiplicar um número inteiro por um número em ponto flutuante sem a necessidade de uma conversão explícita:

```
int numeroInteiro = 5;

float numeroFloat = 2.5;

 float resultado = numeroInteiro * numeroFloat;
```

Nesse exemplo, a multiplicação ocorre entre um inteiro e um ponto flutuante, e C realiza a conversão automática do inteiro para ponto flutuante antes da multiplicação.

Embora essa flexibilidade possa ser conveniente, também pode levar a resultados inesperados quando as conversões automáticas não são desejadas ou não são totalmente compreendidas.

Em resumo, a linguagem de programação C é tipada estaticamente porque verifica tipos em tempo de compilação, mas também é considerada tipagem fraca devido à flexibilidade nas conversões automáticas de tipos, o que pode levar a situações inesperadas se não for usado com cuidado.

5)
```
function passThrough(value: any): any {
    return value;
}
const stringValue: string = "Olá, mundo!";
const result: any = passThrough(stringValue);

console.log(result.toUpperCase());
```

   Neste exemplo, a função passThrough aceita qualquer tipo de parâmetro, e o tipo de retorno também é definido como any. Isso permite que você repasse o valor sem preocupações com a verificação de tipos. Embora o uso do tipo any aqui possa ser benéfico para manter a flexibilidade, é importante notar que a verificação de tipos estática do TypeScript não estará ativa para esse valor. Portanto, você deve ter certeza de que está ciente das possíveis implicações e riscos do uso do any, especialmente em projetos maiores onde a segurança de tipos é importante.

6) Embora um valor do tipo number possa representar tanto números inteiros quanto números de ponto flutuante, essa característica não é indicativa de tipagem fraca, mas sim de tipagem forte com flexibilidade nos tipos numéricos. No TypeScript, mesmo que números inteiros e de ponto flutuante possam ser tratados como number, as operações entre eles ainda são restritas pelas regras de tipos, e o compilador TypeScript verifica a compatibilidade de tipos. Portanto, enquanto o TypeScript permite que números inteiros e de ponto flutuante sejam representados pelo mesmo tipo number, ele ainda é uma linguagem de tipagem estática e forte, enfatizando a segurança de tipos em tempo de compilação. Isso difere das linguagens de tipagem fraca, nas quais as conversões automáticas de tipos ocorrem de forma mais liberal, muitas vezes sem que o programador esteja ciente.

7)

```
class Dados {
   name: string = "Ely";
   paymentTime: number = 120.56;
   language: string = "TypeScript";
}
let dadosInstance = new Dados();
const message = `${dadosInstance.name}
```

```
My payment time is ${dadosInstance.paymentTime}
And my preferred language is ${dadosInstance.language}`;
console.log(message);
```