Nível 1: Estrutura Básica e Fluxo

1. **O Básico:** Crie um bloco `try...catch`. No `try`, tente acessar uma variável que não foi declarada. No `catch`, exiba "Erro capturado!" no console.
2. **O Objeto Error:** Repita o exercício anterior, mas agora no `catch(error)`, exiba a propriedade `error.message` para ver o que o JS diz.
3. **Sempre Executa:** Adicione um bloco `finally` ao exercício anterior que imprima "Fim da execução", independentemente de ter dado erro ou não.
4. **Fluxo Limpo:** Crie um `try...catch...finally` onde o código no `try` **não** falha (ex: `console.log("Sucesso")`). Observe se o `catch` e o `finally` são executados.
5. **Identificando Erros:** No `catch`, use `if (error instanceof ReferenceError)` para imprimir uma mensagem específica apenas se o erro for de referência.

Nível 2: Lançando Erros (Throw) e Validações

6. **Divisão Segura:** Crie uma função `dividir(a, b)`. Se `b` for igual a `0`, use `throw new Error("Divisão por zero não permitida")`. Caso contrário, retorne a divisão.
7. **Apenas Números:** Crie uma função que aceita um argumento. Se o argumento não for do tipo 'number', lance um erro. Teste com um `try...catch`.
8. **Validação de Senha:** Crie uma função `verificarSenha(senha)`. Se a senha tiver menos de 8 caracteres, lance um erro "Senha muito curta".
9. **String vs Objeto:** Em um `try`, lance uma string pura (`throw "Erro string"`). No `catch`, imprima o erro. Note a diferença para quando se lança um `new Error()`.
10. **Validação de Array:** Crie uma função que recebe um array. Se o array estiver vazio, lance um erro. Capture isso externamente.

Nível 3: JSON e Tipos de Erro

11. **JSON Quebrado:** Tente usar `JSON.parse()` em uma string que não é um JSON válido (ex: `"{ nome: 'gui' "`). Capture o `SyntaxError`.
12. **Propriedade de Nulo:** Tente acessar `pessoa.nome` onde `pessoa` é `null`. Capture o `TypeError` resultante.
13. **Erro Personalizado (Classe):** Crie uma classe `ErroDeNegocio` que estende `Error`. Lance uma instância desse erro e capture-a verificando o tipo com `instanceof`.

14. **Aninhamento (Nested):** Coloque um `try...catch` dentro de outro bloco `try`. Force um erro no interno e veja se o externo também é acionado (spoiler: não deveria, se o interno tratar o erro).
15. **Rethrow (Relançar):** No bloco `catch`, capture um erro, faça um `console.log` ("Logando erro...") e depois use `throw error` novamente para que o erro suba para um nível superior.

🟥 Nível 4: Assíncrono e Cenários Reais

16. **Async/Await:** Crie uma função `async`. Dentro dela, use `await` em uma Promise que rejeita (reject). Envolva isso em `try...catch` para capturar o erro da Promise.
17. **Fetch Seguro:** (Simulado) Crie uma função assíncrona que tenta buscar dados de uma URL inexistente (ou simule uma falha). Trate o erro de rede.
18. **Fallbacks:** Crie uma função que tenta ler uma propriedade de um objeto. Se der erro, o `catch` deve retornar um valor padrão (ex: "Valor desconhecido") em vez de travar ou lançar erro novo.
19. **Validação em Lote:** Crie um array de usuários (alguns válidos, outros inválidos). Use um loop `forEach` ou `for...of`. Dentro do loop, use `try...catch` para validar cada usuário individualmente, garantindo que um erro não pare a validação dos próximos.
20. **Simulação de Banco de Dados:** Crie uma função `conectarBanco()`. Use um `Math.random()` para simular 50% de chance de falha. Se falhar, lance erro. Use `try...catch` e tente conectar até conseguir (loop com tentativa) ou até um limite de tentativas.