Vamos criar um exercício prático simples para entender o conceito de *Promises* em *JavaScript*. Suponha que temos uma função baixarConteudoPromisse que simula o *download* de um conteúdo da internet, mas desta vez, utilizaremos *Promises*.

```
JS promisses.js > ...
1   function baixarConteudoPromisse(nomeConteudo) {
2       return new Promise((resolve, reject) => {
3         console.log(`Iniciando o download de ${nomeConteudo}...`);
4
5         // Simula um tempo de download (em milissegundos)
6         setTimeout(() => {
7           const sucesso = true; // Simula se o download foi bem-sucedido
8
9           if (sucesso) {
10            console.log(`${nomeConteudo} foi baixado com sucesso.`);
11            resolve(nomeConteudo); // Resolvendo a Promise com sucesso
12          } else {
13            const erro = 'Erro ao baixar o conteúdo';
14            console.error(erro);
15            reject(erro); // Rejeitando a Promise em caso de erro
16          }
17        }, 2000); // Simula 2 segundos de download
18      });
19    }
20
21    // Vamos usar a função baixarConteudoPromisse com Promises
22    baixarConteudoPromisse('Documento.pdf')
23      .then((nomeConteudo) => {
24        console.log(`Download de ${nomeConteudo} concluído com sucesso!`);
25      })
26      .catch((erro) => {
27        console.error(`Erro durante o download: ${erro}`);
28      });
29
30    // Podemos encadear múltiplos downloads usando .then
31    baixarConteudoPromisse('Imagem.jpg')
32      .then((nomeConteudo) => {
33        console.log(`Download de ${nomeConteudo} concluído com sucesso!`);
34        return baixarConteudoPromisse('Video.mp4');
35      })
36      .then((nomeConteudo) => {
37        console.log(`Download de ${nomeConteudo} concluído com sucesso!`);
38      })
39      .catch((erro) => {
40        console.error(`Erro durante o download: ${erro}`);
41      });
42
```

Neste exemplo, a função baixarConteudoPromisse retorna uma *Promise* que representa o download do conteúdo. A *Promise* é resolvida se o *download* for bem-sucedido e rejeitada se ocorrer algum erro.

Ao usar *.then* e *.catch*, podemos tratar o resultado da *Promise*. Se a *Promise* for resolvida, o código dentro do *.then* é executado. Se a *Promise* for rejeitada, o código dentro do *.catch* é executado.

Além disso, podemos encadear várias *Promises* sequencialmente usando vários blocos *.then*, o que proporciona um código mais limpo e legível para operações assíncronas.