1. **Função com Valores de Fallback**  
   Crie uma função chamada greet que receba dois parâmetros: name e greeting.
   * Se greeting não for passado, ele deve ter um valor padrão de "Hello".
   * A função deve retornar uma string no formato: "greeting, name!".
   * Exemplo de uso:

javascript

Copiar código

console.log(greet("Alice")); // "Hello, Alice!"

console.log(greet("Bob", "Hi")); // "Hi, Bob!"

1. **Uso de arguments**  
   Escreva uma função chamada sumAll que aceita qualquer número de argumentos e retorna a soma de todos eles.
   * Use a palavra-chave arguments para acessar os parâmetros.
   * Exemplo de uso:

javascript

Copiar código

console.log(sumAll(1, 2, 3, 4)); // 10

console.log(sumAll(5, 10)); // 15

1. **Desestruturação de Parâmetros**  
   Crie uma função chamada introduce que receba um objeto como parâmetro com as propriedades name, age, e job.
   * Use a desestruturação para acessar os valores e retorne uma string no formato:  
     "Hi, I'm [name], a [age]-year-old [job]."
   * Exemplo de uso:

javascript

Copiar código

console.log(introduce({ name: "Alice", age: 30, job: "developer" }));

// "Hi, I'm Alice, a 30-year-old developer."

1. **Parâmetros Rest (...args)**  
   Escreva uma função chamada filterEvenNumbers que aceite um número indefinido de argumentos.
   * Retorne um array contendo apenas os números pares.
   * Exemplo de uso:

javascript

Copiar código

console.log(filterEvenNumbers(1, 2, 3, 4, 5, 6)); // [2, 4, 6]

**Questões Abertas**

1. **Diferença entre arguments e ...args**  
   Explique a diferença entre arguments e o operador de parâmetros rest (...args) em funções JavaScript.
   * Qual deles você usaria em uma função de seta (arrow function) e por quê?

A principal diferença entre arguments e ...args está no suporte do uso de cada um, sendo que arguments é suportado apenas no uso de funções declaradas com a palavra reservada “function”, enquanto ...args é suportado por arrow functions. Ambos são recursos disponibilizados pelo Javascript para capturar argumentos passados e armazená-los em um array (na verdade argumentos é um objeto semelhante a um array enquanto ...args é um operador de coleta que armazena os dados em um array verdadeiro.) sem a necessidade de ter parâmetros para recebê-los.

Eu usaria o ...args pois ele é suportado por arrow functions, desde a atualização da ES6, enquanto arguments não é.

**Resposta do chatgpt:**A principal diferença entre `arguments` e `...args` é que `arguments` é um objeto semelhante a um array, disponível apenas em funções tradicionais, enquanto `...args` é um operador que coleta argumentos em um array verdadeiro.

`arguments` não funciona em arrow functions, pois estas não possuem seu próprio contexto de execução (lexical `this`), mas `...args` é suportado em ambos os tipos de função.

Eu usaria `...args` em funções modernas, especialmente em arrow functions, devido à sua flexibilidade e suporte nativo em ES6 e posterior.

1. **Desestruturação em Parâmetros de Função**  
   Considere o código abaixo:

javascript

Copiar código

function displayUser({ name, age = 18 }) {

console.log(`${name} is ${age} years old.`);

}

displayUser({ name: "Alice" });

displayUser({ name: "Bob", age: 25 });

* + O que acontece se a função for chamada como displayUser({})? Ela não dará erro, porem irá exibir em tela `undefined is 18 years old.`
  + E se for chamada como displayUser()? Será exibido um erro em tela de sintaxe, pois não foi passado nada, nem mesmo um objeto vazio.
  + Como você modificaria o código para evitar erros quando a função for chamada sem argumentos? Adicionaria um valor de fallback para o parâmetro “name”.

**Resposta do chat gpt:**

**A sugestão de adicionar fallback está incompleta. É importante mostrar como adicionar o fallback para o objeto inteiro, além de suas propriedades.**

function displayUser({ name = "Anonymous", age = 18 } = {}) {

console.log(`${name} is ${age} years old.`);

}

* Explicação:
  + Ao adicionar = {} ao parâmetro, você evita erros quando a função é chamada sem argumentos.
  + Adicionar valores padrão para as propriedades garante que a função funcione mesmo com um objeto incompleto.

1. **Validação de Parâmetros**  
   Por que é importante validar os parâmetros em uma função?

É importante validar os parâmetros da função para evitar comportamento inesperados e exibição de erros que atrapalhem a experiencia do usuário. Podemos exibir até mesmo o erro para o usuário de uma forma mais sucinta que indique alguma instabilidade temporária, mas sem expor qual exatamente é o erro.

* + Dê um exemplo de uma função que verifica se o argumento passado é do tipo esperado e lança um erro se não for.

function verifyArgument(name) {

try {

if(typeof name === “string”) console.log(name)

}

catch(error) {

throw new SyntaxError tipo esperado não foi informado”, error.message)

}

}

**Reposta do chat gpt:**

* **Explicou bem a importância da validação para evitar comportamentos inesperados e melhorar a experiência do usuário.**
* **Mencionou exibir erros de forma amigável, o que é uma boa prática.**
* **O exemplo fornecido contém problemas de sintaxe e lógica:**
  + **A função verifyArgument usa try/catch de forma inadequada. Não é necessário capturar um erro que não foi lançado.**
  + **A string usada no throw está mal formatada.**

**Resposta Melhorada:**

javascript

Copiar código

function verifyArgument(name) {

if (typeof name !== "string") {

throw new TypeError("O argumento 'name' deve ser uma string.");

}

console.log(name);

}

try {

verifyArgument(123); // Teste com valor inválido

} catch (error) {

console.error(error.message);

}

* **Explicação**:
  + Valide explicitamente o tipo do argumento e lance um erro apropriado (TypeError).
  + Use try/catch apenas ao chamar a função, para capturar e tratar erros.

1. **Funções com Parâmetros Nomeados**  
   Em quais situações você recomendaria o uso de parâmetros nomeados (usando objetos como argumento)?
   * Cite vantagens e desvantagens dessa abordagem em comparação com funções que usam vários parâmetros posicionais.

Em situações que eu precisasse encapsular informações, receber parâmetros nomeados, seria o melhor, pois conseguiria mandar para o banco de dados as informações completas de capa instância. Nesse tipo de abordagem, a grande vantagem é o encapsulamento de informações, quanto às desvantagens, penso que se formos receber dados que não possuem uma ligação, usar parâmetros posicionais pode ser melhor.

**Sua Resposta:**

* **Pontos Fortes**:
  + Identificou que parâmetros nomeados são úteis para encapsular informações.
  + Reconheceu que parâmetros posicionais podem ser mais simples para dados não relacionados.
* **Melhorias**:
  + A explicação pode ser expandida, abordando mais vantagens e desvantagens.

**Resposta Melhorada pelo chatgpt:**

plaintext

Copiar código

Eu recomendaria o uso de parâmetros nomeados em situações onde:

- A função recebe muitos parâmetros, e é importante identificar claramente o propósito de cada um.

- Alguns parâmetros são opcionais, permitindo valores padrão.

- A ordem dos argumentos não é relevante.

\*\*Vantagens:\*\*

- Melhor legibilidade e clareza do código.

- Facilidade para adicionar ou remover parâmetros sem quebrar chamadas existentes.

- Flexibilidade ao passar apenas os argumentos necessários.

\*\*Desvantagens:\*\*

- Pode ser mais verboso em comparação com parâmetros posicionais.

- Exige mais cuidado para desestruturar e validar os parâmetros.

\*\*Exemplo Comparativo:\*\*

```javascript

// Parâmetros Nomeados

function createUser({ name, age, email }) {

console.log(`Name: ${name}, Age: ${age}, Email: ${email}`);

}

// Parâmetros Posicionais

function createUserPositional(name, age, email) {

console.log(`Name: ${name}, Age: ${age}, Email: ${email}`);

}

// Chamada

createUser({ name: "Alice", age: 25, email: "alice@example.com" });

createUserPositional("Alice", 25, "alice@example.com");

**Nova revisão**

1. **Validação de Tipos e Lançamento de Erros** Crie uma função chamada calculateArea que receba dois parâmetros: length e width.
   * A função deve lançar um erro se qualquer um dos parâmetros não for um número ou for menor ou igual a zero.
   * Caso os parâmetros sejam válidos, a função deve retornar a área do retângulo.

**Exemplo de saída:**

javascript

CopiarEditar

calculateArea(5, 10); // Retorna 50

calculateArea(-2, 10); // Lança um erro: "Os parâmetros devem ser números positivos."

calculateArea("5", 10); // Lança um erro: "Os parâmetros devem ser números."

1. **Parâmetros Nomeados e Fallback com Lançamento de Erros** Escreva uma função chamada createUserProfile que receba um objeto como parâmetro com as propriedades:
   * name (obrigatório, string)
   * age (opcional, número com valor padrão de 18)
   * email (obrigatório, string)

A função deve:

* + Lançar um erro se name ou email não forem fornecidos ou não forem do tipo string.
  + Retornar um objeto contendo as informações fornecidas.

**Exemplo de saída:**

javascript

CopiarEditar

createUserProfile({ name: "Alice", email: "alice@example.com" });

// Retorna: { name: "Alice", age: 18, email: "alice@example.com" }

createUserProfile({ name: "Bob", email: 123 });

// Lança um erro: "Email deve ser uma string."

**Questões Abertas**

1. **Por que usar validação de parâmetros em funções?**
   * Explique a importância de validar parâmetros em funções.
   * Quais são os impactos de não validar os argumentos recebidos?
   * Você prefere lançar erros ou usar valores padrão para lidar com argumentos inválidos? Por quê?

É importante realizar a validação de parâmetros nas funções para evitar comportamentos indevidos, como recebimento de tipo errado e valores não enviados pelo usuário por exemplo. Se não é realizada uma validação de parâmetros a função pode ter retornos confusos ao usuário, o que dificulta sua experiência, por exemplo, em uma função de cálculo de imc, se a altura e peso não forem validados e o usuário passar valores negativos, a função irá funcionar, mas o retorno será um valor errôneo.

Antes eu usava apenas valores de fallback porque achava mais fácil para lidar com erros, mas hoje uso lançamento de erros na maioria das vezes, pois lançar um erro informando que algo não está corretamente passado ao usuário, assegura um melhor funcionamento do sistema e assegura a integridade dos dados.

1. **Parâmetros Rest vs. Spread Operator**
   * Qual é a diferença entre o operador de parâmetros rest (...args) e o operador de espalhamento (...spread)?
   * Cite um exemplo prático para cada caso de uso e explique como eles ajudam no desenvolvimento de funções mais flexíveis.

O rest operator, como o próprio nome sugere, é um operador que permite pegar o resto dos elementos seja de um array ou de parâmetros passados e criar um vetor contendo eles.

Já o spread operator, trabalha para espalhar valores, imagino. Eu não sei muito sobre ele.

No caso do rest operator, pode ser útil para por exemplo armazenar um número indeterminado (não se sabe quantos parâmetros vão ser passados) de valores e após isso, realizarmos a soma:

const acumulator = (...args) => {

    let total = 0;

    for(let arg of args) {

        total += arg;

    }

    return total;

}

console.log(acumulator(13, 45, 1, 0, 3, 4, 5));

No caso do spread operator, eu não sei.

**2. Spread Operator:**

O **spread operator** é usado para espalhar os elementos de um array ou objeto. Ele pode ser útil para copiar arrays, combinar arrays, ou passar os elementos de um array como argumentos para uma função.

Exemplo de uso com arrays:

javascript

CopiarEditar

const numbers = [1, 2, 3, 4];

const moreNumbers = [5, 6, 7, 8];

// Usando spread para combinar arrays

const combined = [...numbers, ...moreNumbers];

console.log(combined); // Resultado: [1, 2, 3, 4, 5, 6, 7, 8]

Exemplo de uso com objetos:

javascript

CopiarEditar

const person = { name: "Alice", age: 25 };

const updatedPerson = { ...person, job: "Developer" };

console.log(updatedPerson); // Resultado: { name: "Alice", age: 25, job: "Developer" }

**Resumindo:**

* **Rest (...args)**: Captura um número indefinido de parâmetros em uma função, colocando-os em um array.
* **Spread (...spread)**: Espalha os elementos de um array ou objeto, permitindo que você os passe como elementos individuais.

**Como eles ajudam no desenvolvimento de funções mais flexíveis:**

* O **rest operator** permite que você crie funções que aceitam um número variável de parâmetros, sem precisar especificar quantos parâmetros a função pode ter.
* O **spread operator** facilita a manipulação de arrays e objetos, tornando o código mais limpo e flexível ao combinar ou passar dados de maneira mais eficiente.

Agora você tem uma explicação clara de como os dois operadores funcionam e como usá-los! 😊