Questões de Imutabilidade
FÁCEIS

1- Qual método de array JavaScript **NÃO** altera o array original e, portanto, é considerado imutável?
a) splice()
b) map()
c) sort()
d) reverse()

Resposta: b) map()

2-Qual das seguintes estruturas em JavaScript é imutável por padrão?
a) Arrays
b) Objetos
c) Strings
d) Mapas

Resposta: c) Strings

3-Em qual dos cenários abaixo a imutabilidade é fundamental para evitar problemas?
a) Ao alterar o estado global da aplicação
b) Ao trabalhar com dados de um banco de dados
c) Ao trabalhar com funções puras em programação funcional
d) Ao criar interfaces de usuário com for loops

Resposta: c) Ao trabalhar com funções puras em programação funcional

4- Qual dos métodos a seguir cria uma cópia imutável de um objeto em JavaScript?
a) Object.assign()
b) Object.freeze()
c) Object.seal()
d) delete

Resposta: b) Object.freeze()

5-No contexto de programação funcional, qual é a principal vantagem da imutabilidade?
a) Melhor desempenho
b) Facilita a depuração e o teste
c) Menos consumo de memória
d) Código mais curto

Resposta: b) Facilita a depuração e o teste

6-O que acontece se você tentar modificar uma propriedade de um objeto congelado usando Object.freeze()?
a) O valor será alterado
b) Uma exceção será lançada
c) O valor permanecerá o mesmo
d) O objeto será excluído

Resposta: c) O valor permanecerá o mesmo

7-Qual das alternativas a seguir é uma técnica comum em programação funcional para garantir a imutabilidade de objetos?
a) Modificar o objeto diretamente
b) Clonar o objeto antes de modificar
c) Usar loops for para iterar
d) Usar funções recursivas

Resposta: b) Clonar o objeto antes de modificar

8-Qual das funções de array abaixo é imutável?
a) push()
b) splice()
c) filter()
d) reverse()

Resposta: c) filter()

9-Em programação funcional, por que a imutabilidade é importante ao trabalhar com funções puras?
a) Porque funções puras não podem retornar valores
b) Porque garante que o estado não seja alterado inesperadamente
c) Porque funções puras sempre modificam os argumentos
d) Porque funções puras podem ter efeitos colaterais

Resposta: b) Porque garante que o estado não seja alterado inesperadamente

10-Qual das alternativas demonstra uma abordagem funcional e imutável para atualizar o valor de uma propriedade em um objeto?
a) obj.name = "Novo Nome"
b) Object.assign(obj, { name: "Novo Nome" })
c) const newObj = { ...obj, name: "Novo Nome" }
d) delete obj.name

Resposta: c) const newObj = { ...obj, name: "Novo Nome" }

MÉDIAS

1-Qual das seguintes alternativas é uma prática comum em programação funcional para evitar mutabilidade?
a) Usar variáveis var
b) Usar let para redefinir o valor de uma variável
c) Usar const sempre que possível
d) Modificar diretamente objetos globais

Resposta: c) Usar const sempre que possível

2-O que significa "imutabilidade profunda" em JavaScript?
a) Não permitir alterações em qualquer nível de um objeto, incluindo objetos aninhados
b) Garantir que apenas a primeira camada de propriedades de um objeto seja imutável
c) Clonar um objeto e suas propriedades mais internas
d) Remover todas as referências de um objeto

Resposta: a) Não permitir alterações em qualquer nível de um objeto, incluindo objetos aninhados

3-Qual das opções abaixo é uma maneira de garantir a imutabilidade de objetos complexos em JavaScript?
a) Usar loops para iterar e modificar o objeto
b) Usar Object.freeze() em todos os níveis do objeto
c) Redefinir o objeto com delete
d) Usar splice() para modificar arrays aninhados

Resposta: b) Usar Object.freeze() em todos os níveis do objeto

4-Qual das alternativas a seguir NÃO é considerada uma técnica de imutabilidade?
a) Usar concat() para adicionar elementos a um array
b) Usar map() para modificar arrays
c) Usar splice() para remover itens de um array
d) Usar o operador de espalhamento (spread) para copiar objetos

Resposta: c) Usar splice() para remover itens de um array

5-Qual das opções a seguir exemplifica uma maneira de evitar a mutação de arrays?
a) array.push(item)
b) array.splice(index, 1)
c) array.filter(item => item !== value)
d) array.reverse()

Resposta: c) array.filter(item => item !== value)

6-Qual é a diferença entre imutabilidade superficial e imutabilidade profunda?
a) Imutabilidade superficial se aplica a arrays, e imutabilidade profunda a objetos
b) Imutabilidade superficial congela apenas o primeiro nível, enquanto imutabilidade profunda congela todos os níveis
c) Imutabilidade profunda permite modificar propriedades internas
d) Não há diferença, ambos os termos são intercambiáveis

Resposta: b) Imutabilidade superficial congela apenas o primeiro nível, enquanto imutabilidade profunda congela todos os níveis

7-Qual das opções a seguir exemplifica uma maneira de evitar a mutação de arrays?
a) array.push(item)
b) array.splice(index, 1)
c) array.filter(item => item !== value)
d) array.reverse()

Resposta: c) array.filter(item => item !== value)

8-Qual é o principal benefício de garantir a imutabilidade em funções assíncronas?
a) O código se torna mais difícil de manter
b) O código pode ser executado mais rapidamente
c) O código fica mais previsível, pois elimina problemas de concorrência
d) A imutabilidade não tem impacto em funções assíncronas

Resposta: c) O código fica mais previsível, pois elimina problemas de concorrência

9-Qual é a principal desvantagem do uso de imutabilidade em estruturas de dados complexas?
a) A imutabilidade aumenta o risco de erros no código
b) A imutabilidade aumenta a complexidade do gerenciamento de estado
c) A imutabilidade pode aumentar o uso de memória, já que cópias são criadas
d) A imutabilidade torna o código menos eficiente

Resposta: c) A imutabilidade pode aumentar o uso de memória, já que cópias são criadas

10-Qual das opções a seguir é uma maneira imutável de remover uma propriedade de um objeto em JavaScript?
a) delete obj.prop
b) const newObj = Object.assign({}, obj); delete newObj.prop;
c) obj.prop = undefined
d) obj = { ...obj, prop: null }

Resposta: b) const newObj = Object.assign({}, obj); delete newObj.prop;

DIFÍCEIS

1-Quando se trabalha com referência imutável em JavaScript, o que pode ser alterado em um objeto declarado com const?
a) O valor da referência
b) As propriedades internas do objeto
c) Nenhuma parte do objeto pode ser alterada
d) Apenas arrays podem ser alterados, não objetos

Resposta correta: b) As propriedades internas do objeto

2-Qual das alternativas a seguir é verdadeira sobre a imutabilidade estrutural?
a) Toda modificação em um objeto resulta em uma cópia completamente nova, mesmo para objetos aninhados
b) Apenas o primeiro nível de propriedades é copiado, preservando as referências internas
c) Imutabilidade estrutural sempre é garantida em JavaScript por padrão
d) A imutabilidade estrutural reduz o consumo de memória, copiando apenas as propriedades que mudaram

Resposta correta: d) A imutabilidade estrutural reduz o consumo de memória, copiando apenas as propriedades que mudaram

3-Qual das bibliotecas abaixo é comumente usada para garantir imutabilidade estrutural em JavaScript?
a) Lodash
b) Ramda
c) Immutable.js
d) jQuery

Resposta correta: c) Immutable.js

4-Considere o seguinte código:
```
const obj1 = { a: 1, b: { c: 2 } };
const obj2 = { ...obj1 };
obj2.b.c = 3;
```
Por que obj1.b.c também é modificado? a) Porque o operador de espalhamento faz uma cópia profunda
b) Porque obj2 é uma referência para obj1
c) Porque o operador de espalhamento faz uma cópia rasa (shallow copy)
d) Porque const foi usado para declarar obj1

Resposta correta: c) Porque o operador de espalhamento faz uma cópia rasa (shallow copy)

5-Qual é a principal vantagem de usar estruturas de dados persistentes em comparação com estruturas mutáveis?
a) Aumentar a capacidade de manipulação direta de memória
b) Permitir que dados antigos sejam descartados automaticamente
c) Melhorar a performance de operações de entrada e saída (I/O)
d) Evitar bugs relacionados à mutação e melhorar a previsibilidade do código

Resposta correta: d) Evitar bugs relacionados à mutação e melhorar a previsibilidade do código

6-O que torna o método Array.prototype.map() uma função de ordem superior adequada para programação funcional e imutabilidade?
a) Modifica o array original
b) Retorna um novo array sem alterar o original
c) Depende de variáveis globais
d) Substitui os valores do array original por novos valores

Resposta correta: b) Retorna um novo array sem alterar o original

7-Considere o seguinte código:

```
const obj1 = { a: 1, b: { c: 2 } };
const obj2 = JSON.parse(JSON.stringify(obj1));
obj2.b.c = 3;
```

Por que o valor de obj1.b.c não foi alterado? a) O operador de espalhamento foi utilizado
b) Foi realizada uma cópia rasa (shallow copy)
c) Foi realizada uma cópia profunda (deep copy)
d) Ambos obj1 e obj2 são referências ao mesmo objeto

Resposta correta: c) Foi realizada uma cópia profunda (deep copy)

8-Considere a seguinte função em JavaScript:

```
function addProperty(obj, key, value) {
  const newObj = { ...obj, [key]: value };
  return newObj;
}
```

Qual das opções descreve corretamente o comportamento desta função? a) Ela modifica diretamente o objeto original
b) Ela copia o objeto original e retorna uma versão modificada
c) Ela retorna uma cópia rasa (shallow copy) sem a nova propriedade
d) Ela realiza uma cópia profunda (deep copy) do objeto original

Resposta correta: b) Ela copia o objeto original e retorna uma versão modificada

9-Qual é o comportamento do currying em programação funcional, relacionado à imutabilidade?
a) Currying transforma uma função que recebe múltiplos argumentos em uma função que recebe um único argumento de cada vez, sem modificar variáveis externas
b) Currying cria funções que alteram os argumentos antes de executá-las
c) Currying é um padrão que permite modificar o estado de variáveis internas da função
d) Currying cria funções mutáveis com base em parâmetros dinâmicos

Resposta correta: a) Currying transforma uma função que recebe múltiplos argumentos em uma função que recebe um único argumento de cada vez, sem modificar variáveis externas

10- O que significa que um sistema usa "Imutabilidade transiente" em programação funcional?
a) Um estado imutável que pode ser temporariamente mutável durante cálculos intermediários
b) Um estado que muda de imutável para mutável após um determinado tempo
c) Um estado que é imutável em um escopo e mutável em outro
d) Um estado que se mantém imutável apenas em linguagens com tipagem estática

Resposta correta: a) Um estado imutável que pode ser temporariamente mutável durante cálculos intermediários