Exercice 6 - Optimisation avec useMemo et useCallback

Objectif

Comprendre et appliquer les hooks d'optimisation pour éviter les re-renders inutiles.

Concepts abordés

- useMemo pour mémoriser des valeurs calculées
- useCallback pour mémoriser des fonctions
- React.memo pour éviter les re-renders de composants
- Quand (ne pas) optimiser

Quand utiliser ces hooks ?

Hook	Utilisation
`useMemo`	Calculs coûteux, valeurs dérivées complexes
`useCallback`	Fonctions passées en props à des composants mémoïsés
`React.memo`	Composants qui reçoivent les mêmes props souvent

⚠ Règle d'or

> **N'optimisez pas prématurément !** Ces hooks ont un coût. Utilisez-les uniquement si vous constatez un problème de performance.

Énoncé

Créer une application de liste de produits avec :

1. Une liste de 1000 produits
2. Un filtre de recherche
3. Un tri (prix, nom)
4. Un compteur indépendant

Optimiser pour que :

- Le compteur ne re-calcule pas la liste filtrée
- Les composants enfants ne re-rendent pas inutilement

Correction

```jsx
import { useState, useMemo, useCallback, memo } from 'react';
import './ProductList.css';

// Générer des produits fictifs
const generateProducts = (count) => {
  return Array.from({ length: count }, (_, i) => ({
    id: i + 1,
    name: `Produit ${i + 1}`,
    price: Math.floor(Math.random() * 1000) + 10,
    category: ['Tech', 'Mode', 'Maison', 'Sport'][Math.floor(Math.random() * 4)]
  }));
};

const ALL_PRODUCTS = generateProducts(1000);

// ═══════════════════════════════════════════════════════════════════════
// COMPOSANT ENFANT MÉMOÏSÉ
// React.memo empêche le re-render si les props n'ont pas changé
// ═══════════════════════════════════════════════════════════════════════
const ProductCard = memo(function ProductCard({ product, onAddToCart }) {
  console.log(`Render ProductCard ${product.id}`);

  return (
    <div className="product-card">
      <h3>{product.name}</h3>
      <p className="price">{product.price} €</p>
      <span className="category">{product.category}</span>
      <button onClick={() => onAddToCart(product.id)}>
        Ajouter au panier
      </button>
    </div>
  );
});

// ═══════════════════════════════════════════════════════════════════════
// COMPOSANT PRINCIPAL
// ═══════════════════════════════════════════════════════════════════════
function ProductList() {
  const [search, setSearch] = useState('');
  const [sortBy, setSortBy] = useState('name');
  const [count, setCount] = useState(0);
  const [cart, setCart] = useState([]);

  // ─────────────────────────────────────────────────────────────────────
  // useMemo : Mémoriser le résultat d'un calcul coûteux
  //
  // Sans useMemo, ce calcul serait refait à CHAQUE render,
  // même quand on incrémente le compteur (qui n'affecte pas la liste)
  //
  // Avec useMemo, le calcul n'est refait que si search ou sortBy change
  // ─────────────────────────────────────────────────────────────────────
  const filteredAndSortedProducts = useMemo(() => {
    console.log('🔄 Calcul de la liste filtrée...');
```

```
    // Filtrer
    let result = ALL_PRODUCTS.filter(p =>
      p.name.toLowerCase().includes(search.toLowerCase())
    );

    // Trier
    result = [...result].sort((a, b) => {
      if (sortBy === 'name') return a.name.localeCompare(b.name);
      if (sortBy === 'price-asc') return a.price - b.price;
      if (sortBy === 'price-desc') return b.price - a.price;
      return 0;
    });

    return result;
  }, [search, sortBy]); // Dépendances : recalculer si ces valeurs changent

  // ─────────────────────────────────────────────────────────────
  // useCallback : Mémoriser une fonction
  //
  // Sans useCallback, handleAddToCart serait recréé à chaque render,
  // ce qui causerait le re-render de TOUS les ProductCard (même avec memo)
  //
  // Avec useCallback, la même référence de fonction est réutilisée
  // ─────────────────────────────────────────────────────────────
  const handleAddToCart = useCallback((productId) => {
    setCart(prev => [...prev, productId]);
    console.log(`Produit ${productId} ajouté au panier`);
  }, []); // Pas de dépendances : la fonction ne change jamais

  // ─────────────────────────────────────────────────────────────
  // useMemo pour une valeur dérivée simple
  // ─────────────────────────────────────────────────────────────
  const totalInCart = useMemo(() => {
    return cart.reduce((sum, productId) => {
      const product = ALL_PRODUCTS.find(p => p.id === productId);
      return sum + (product?.price || 0);
    }, 0);
  }, [cart]);

  return (
    <div className="product-list-container">
      {/* Compteur indépendant - ne devrait pas recalculer la liste */}
      <div className="counter-section">
        <p>Compteur : {count}</p>
        <button onClick={() => setCount(c => c + 1)}>+1</button>
        <p className="hint">
          Cliquez et regardez la console : la liste n'est pas recalculée !
        </p>
      </div>

      {/* Panier */}
      <div className="cart-section">
        <p>🛒 Panier : {cart.length} articles - Total : {totalInCart} €</p>
```

```
      </div>

      {/* Filtres */}
      <div className="filters">
        <input
          type="text"
          placeholder="Rechercher..."
          value={search}
          onChange={e => setSearch(e.target.value)}
        />
        <select value={sortBy} onChange={e => setSortBy(e.target.value)}>
          <option value="name">Nom</option>
          <option value="price-asc">Prix croissant</option>
          <option value="price-desc">Prix décroissant</option>
        </select>
      </div>

      {/* Résultats */}
      <p className="results-count">
        {filteredAndSortedProducts.length} produits trouvés
      </p>

      {/* Liste (limité à 50 pour l'affichage) */}
      <div className="products-grid">
        {filteredAndSortedProducts.slice(0, 50).map(product => (
          <ProductCard
            key={product.id}
            product={product}
            onAddToCart={handleAddToCart}
          />
        ))}
      </div>
    </div>
  );
}

export default ProductList;
```

```
/* ProductList.css */
.product-list-container {
  padding: 1rem;
  max-width: 1200px;
  margin: 0 auto;
}

.counter-section, .cart-section {
  padding: 1rem;
  background: #f3f4f6;
  border-radius: 8px;
  margin-bottom: 1rem;
}
```

```css
.hint {
  font-size: 0.75rem;
  color: #6b7280;
}

.filters {
  display: flex;
  gap: 1rem;
  margin-bottom: 1rem;
}

.filters input, .filters select {
  padding: 0.5rem;
  border: 1px solid #e5e7eb;
  border-radius: 4px;
  font-size: 1rem;
}

.filters input {
  flex: 1;
}

.results-count {
  color: #6b7280;
  margin-bottom: 1rem;
}

.products-grid {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
  gap: 1rem;
}

.product-card {
  padding: 1rem;
  border: 1px solid #e5e7eb;
  border-radius: 8px;
  background: white;
}

.product-card h3 {
  margin: 0 0 0.5rem;
  font-size: 1rem;
}

.product-card .price {
  font-weight: bold;
  color: #059669;
  margin: 0;
}

.product-card .category {
  display: inline-block;
```

```
  padding: 0.25rem 0.5rem;
  background: #e5e7eb;
  border-radius: 4px;
  font-size: 0.75rem;
  margin: 0.5rem 0;
}

.product-card button {
  width: 100%;
  padding: 0.5rem;
  background: #3b82f6;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}
```

Schéma : Quand ça recalcule ?

```
Sans useMemo/useCallback :
┌─────────────────────────────────────────────────────────────┐
│  count change → Re-render → Recalcul liste → Re-render enfants  │
│  search change → Re-render → Recalcul liste → Re-render enfants │
│  Tout recalcule à chaque changement !                       │
└─────────────────────────────────────────────────────────────┘

Avec useMemo/useCallback :
┌─────────────────────────────────────────────────────────────┐
│  count change → Re-render → Liste en cache → Enfants en cache   │
│  search change → Re-render → Recalcul liste → Enfants en cache  │
│  Seul ce qui doit changer est recalculé !                   │
└─────────────────────────────────────────────────────────────┘
```