# Understanding useDiscountCalculator and useProfitCalculator Hooks

## Part 1: How useDiscountCalculator Works

This section explains how the custom hook `useDiscountCalculator` performs tiered discount logic based on a user-supplied amount.

▶️ 1. Hook Setup

import { useMemo } from 'react';  
  
function useDiscountCalculator(amount) {  
 const result = useMemo(() => {  
 let discount = 0;  
 if (amount > 1000) {  
 discount = 0.2;  
 } else if (amount > 500) {  
 discount = 0.1;  
 } else if (amount > 100) {  
 discount = 0.05;  
 }  
 const discountedAmount = amount - amount \* discount;  
 return {  
 originalAmount: amount,  
 discountRate: discount,  
 discountedAmount: parseFloat(discountedAmount.toFixed(2))  
 };  
 }, [amount]);  
  
 return result;  
}

This hook uses `useMemo` to avoid unnecessary recalculations. It returns an object containing the original amount, the discount rate, and the final discounted amount.

▶️ 2. Hook Invocation in App.js

const [amount, setAmount] = useState(0);  
const discountData = useDiscountCalculator(amount);

`amount` is stored in local state and passed to the hook. The hook returns a `discountData` object that can be used directly in the JSX.

▶️ 3. UI Rendering

<p>Original: ${discountData.originalAmount}</p>  
<p>Discount: {discountData.discountRate \* 100}%</p>  
<p>Final: ${discountData.discountedAmount}</p>

These values update live when the user changes the input.

## Part 2: How useDiscountCalculator and useProfitCalculator Work Together

This section focuses on how the `useDiscountCalculator` and `useProfitCalculator` hooks work in combination to first calculate a discounted sale price, and then compute profit and margin based on a fixed cost.

▶️ 1. Hook Imports and Initialization

import useDiscountCalculator from './hooks/useDiscountCalculator';  
import useProfitCalculator from './hooks/useProfitCalculator';  
  
const discountData = useDiscountCalculator(amount);  
const profitData = useProfitCalculator(1200, discountData.discountedAmount);

The `discountData.discountedAmount` is passed directly into `useProfitCalculator` along with a fixed cost price of $1200.

▶️ 2. Hook Definition: useProfitCalculator

import { useMemo } from 'react';  
  
function useProfitCalculator(cost, salePrice) {  
 const result = useMemo(() => {  
 const profit = salePrice - cost;  
 const marginPercent = cost > 0 ? ((profit / cost) \* 100).toFixed(2) : 0;  
 return {  
 cost,  
 salePrice,  
 profit: profit.toFixed(2),  
 marginPercent  
 };  
 }, [cost, salePrice]);  
  
 return result;  
}

This hook uses `useMemo` to calculate the profit and margin based on the cost and final sale price. It returns a detailed object containing the financial breakdown.

▶️ 3. Rendering the Profit Data in App.js

<p>Cost Price: $1200</p>  
<p>Sale Price (after discount): ${discountData.discountedAmount}</p>  
<p>Profit: ${profitData.profit}</p>  
<p>Margin: {profitData.marginPercent}%</p>

The profit and margin dynamically update based on the sale price calculated by `useDiscountCalculator`.

▶️ 4. Summary of Workflow

• The user inputs a purchase amount.  
• `useDiscountCalculator` computes the discount and returns a discounted amount.  
• This discounted amount is passed into `useProfitCalculator`, along with a fixed cost.  
• The hook calculates profit and margin and returns that data.  
• The UI reflects both the discount and the profit calculation live.

that part at the end of the hook:

js

CopyEdit

}, [cost, salePrice]);

is part of **React's useMemo dependency array** — and understanding this is key to mastering custom hooks and performance optimization.

Let’s break it down:

**🧠 What is useMemo doing?**

js

CopyEdit

const result = useMemo(() => {

// Some expensive or critical calculation

}, [dependencies]);

useMemo only **recalculates** the value **when one of the dependencies has changed**.

If none of the values in the dependency array change between renders, React **reuses the previously memoized value** — this avoids unnecessary computations.

**📌 So what are these [cost, salePrice] and [amount]?**

**In useDiscountCalculator:**

js

CopyEdit

useMemo(() => {

// calculate discount

}, [amount]);

* amount is the **only input** to the logic.
* If amount hasn’t changed, no need to recalculate discount.
* So, amount is the only dependency.

**In useProfitCalculator:**

js

CopyEdit

useMemo(() => {

// calculate profit = salePrice - cost

}, [cost, salePrice]);

* Both cost and salePrice are **inputs** to the profit calculation.
* If **either** of them changes, we need to recompute the profit.
* So, both are listed in the dependency array.

**⚠️ Why this matters**

React relies on the dependency array to know **when to rerun the memoized function**. If you leave something out:

* You risk using **stale data**.
* Your hook may return the wrong value without re-computing it.

If you include **too many** things:

* The function may rerun more than necessary.
* (Not dangerous, but can impact performance.)

**✅ Rule of Thumb:**

Your dependency array should include **every variable** that is **used inside the memoized function** but **comes from outside it.**

**Examples:**

| **Used inside useMemo** | **Dependency needed?** |
| --- | --- |
| amount | ✅ [amount] |
| cost, salePrice | ✅ [cost, salePrice] |
| Constants (e.g. 0.2) | ❌ Not needed |

A screenshot of a calculator

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.