**COR31TC2NewLoanSetup\_1:**

//#########################################################################################################################################################################################################

//TEST SCRIPT NAME : COR31TC2NewLoanSetup\_1

//OBJECTIVE : Verify the ability to create backdated General Loans for a Direct Facility in Servicing application.

public void COR31TC2NewLoanSetup\_1()

{

WindowsApplication appHandle;

appHandle = ApplicationHandlerFactory.GetApplication(ApplicationType.Windows);

ACBSUtility.ACBSUtilityLibrary.GetCurrentServicingQueueName("GLOBAL\_SERVICINGQUEUE4\_NAME", "YES");

ACBSUtility.ACBSUtilityLibrary.GetCurrentPortFolioName("GLOBAL\_PORTFOLIO4\_NAME", "YES");

string CalenderCode = ACBSUtility.ACBSUtilityLibrary.GetCurrentCalendarID("GLOBAL\_CALENDAR4\_ID", "YES");

ACBSUtility.ACBSUtilityLibrary.GetCurrentRateIndexID("GLOBAL\_RATE\_INDEX4\_ID", "YES");

string Calendar = ACBSUtility.ACBSUtilityLibrary.GetCurrentCalendarName("GLOBAL\_CALENDAR4\_NAME", "YES");

string AccrualScheduleIndexRateTable = ACBSUtility.ACBSUtilityLibrary.GetCurrentRateIndexName("GLOBAL\_RATE\_INDEX4\_NAME", "YES");

ACBSUtility.ACBSUtilityLibrary.CloseAllServicingSubWindows();

string CIF1 = Data.Fetch("NewCustomerSetup", "CustomerNumber");

string FCT1 = Data.Fetch("LIBORP4NewDirectFacilitySetup", "NewBackDatedDirectFacSetup1\_FacilityNumber");

string FacilityEffectiveDate = Data.Fetch("LIBORP4NewDirectFacilitySetup", "NewBackDatedDirectFacSetup1\_FacilityEffectiveDate");

string CurrentSystemDate = Data.Fetch("ServicingLogin", "ApplicationProcessDate");

//----------------------------Script starts from here-----------------------------------------------------------------------------------------

Report.Step("Bring Direct Facility <FCT1> into session. Create 1 month back dated Loan - LN1 by entering all field values (From Loan Servicing Queue window, right click on Facility| Special Actions| Process New| New Loan). Ensure to add a RFR Accrual Schedule (Rate Calculation Method: BASE + SPREAD+ RESERVE, Year Basis: Actual/360, Index Rate Table: <INDEXRATETABLE1>, RFR Calculation Method: SIMPLE, Lag Days: 15) and Primary Repayment Schedule (Bill Type: INTEREST ONLY, Next Due Date: <Loan Effective Date + 2 Months>, Business Day Adjustment: Modified Following, Bill Frequency: MONTHLY, Bill Calendar: <RFRCAL1>, Lead Days: 15).");

string[] Loan1NewLoantabInfo = InputData.Get("COR31TC2BackDatedDirectCreditLoanNewLoanTabFieldValues");

string LoanEffectiveDate = FacilityEffectiveDate;

LoanEffectiveDate = ACBSUtility.ACBSUtilityLibrary.GetBusinessDate(LoanEffectiveDate, "NEXT");

Loan1NewLoantabInfo[7] = LoanEffectiveDate;

Loan1NewLoantabInfo[8] = LoanEffectiveDate;

string[] Loan1AccrualScheduleBasicInformationFieldValues = InputData.Get("COR31TC2BackDatedDirectCreditLoanAccrualScheduleBasicInformationFieldValues");

string LoanEffectiveDatePlus2Months = ACBSUtility.ACBSUtilityLibrary.CalculateNewDate(LoanEffectiveDate, "M", 2);

string NextRatePeriodDate = ACBSUtility.ACBSUtilityLibrary.ConvertDateInGivenFormat(LoanEffectiveDatePlus2Months, "M/dd/yy");

NextRatePeriodDate = ACBSUtility.ACBSUtilityLibrary.GetBusinessDate(NextRatePeriodDate, "NEXT");

Loan1AccrualScheduleBasicInformationFieldValues[19] = AccrualScheduleIndexRateTable;

string[] Loan1RepaymentScheduleBasicInformationFieldValues = InputData.Get("COR31TC2BackDatedDirectCreditLoanRepaymentScheduleBasicInformationFieldValues");

string NextDueDate = NextRatePeriodDate;

Loan1RepaymentScheduleBasicInformationFieldValues[4] = NextDueDate;

Loan1RepaymentScheduleBasicInformationFieldValues[8] = Calendar;

Loan1RepaymentScheduleBasicInformationFieldValues[9] = NextDueDate;

string LN1 = Application.ACBSServicingLibrary.CreateLoan(CIF1, FCT1, Loan1NewLoantabInfo, InputData.Get("COR31TC2BackDatedDirectCreditLoanServicingTabFieldValues"), InputData.Get("COR31TC2BackDatedDirectCreditLoanRiskTabFieldValues"), InputData.Get("COR31TC2BackDatedDirectCreditLoanRegulatoryTabFieldValues"), InputData.Get("COR31TC2BackDatedDirectCreditLoanIndustryDetailsFieldValues"), InputData.Get("COR31TC2BackDatedDirectCreditLoanMiscellaneousTabFieldValues"), null, null, null, null, InputData.Get("COR31TC2BackDatedDirectCreditLoanAccrualScheduleInformationFieldValues"), Loan1AccrualScheduleBasicInformationFieldValues, null, null, InputData.Get("COR31TC2BackDatedDirectCreditLoanRepaymentScheduleInformationFieldValues"), Loan1RepaymentScheduleBasicInformationFieldValues, InputData.Get("COR31TC2BackDatedDirectCreditLoanRepaymentScheduleBillingInformationFieldValues"), null, null, null, null, null, null, null);

string Loan1AccId = RunTimeData.Get("LoanAccrualScheduleID");

Report.Step("Convert the Date in YYYYMMDD format and set the current month end Date as Non Business Day in Database.");

string MonthEndDate = ACBSUtility.ACBSUtilityLibrary.GetMonthEndDate(CurrentSystemDate);

string MonthEndDateFormat = ACBSUtility.ACBSUtilityLibrary.ConvertDateInGivenFormat(MonthEndDate, "yyyyMMdd");

Application.ACBSDBLibrary.SetSpecifiedDateStatusInCalendar(CalenderCode, MonthEndDateFormat, "N");

Report.Step("Convert the Date in YYYYMMDD format and set the future month (Next Month) 1st two day's to Non Business day's in Database.");

string NextMonthFirstDate = ACBSUtility.ACBSUtilityLibrary.CalculateNewDate(MonthEndDate, "D", 1);

string GetMonthFirstDate = ACBSUtility.ACBSUtilityLibrary.GetMonthFirstDate(NextMonthFirstDate);

string NextMonthSecondDate = ACBSUtility.ACBSUtilityLibrary.CalculateNewDate(GetMonthFirstDate, "D", 1);

string GetMonthFirstDateFormat = ACBSUtility.ACBSUtilityLibrary.ConvertDateInGivenFormat(GetMonthFirstDate, "yyyyMMdd");

string NextMonthSecondDateFormat = ACBSUtility.ACBSUtilityLibrary.ConvertDateInGivenFormat(NextMonthSecondDate, "yyyyMMdd");

Application.ACBSDBLibrary.SetSpecifiedDateStatusInCalendar(CalenderCode, GetMonthFirstDateFormat, "N");

Application.ACBSDBLibrary.SetSpecifiedDateStatusInCalendar(CalenderCode, NextMonthSecondDateFormat, "N");

Report.Step("Calculate Loan <LN1> Invoice Print Date.");

string LeadDays = Loan1RepaymentScheduleBasicInformationFieldValues[12];

string LoanInvoicePrintDate = ACBSUtility.ACBSUtilityLibrary.CalculateNewDateWithLagDays(NextDueDate, int.Parse(LeadDays));

ACBSUtility.ACBSUtilityLibrary.Info("Save the Loan Number, Loan Effective Date, Loan Rate Maturity Date, Loan Maturity Date, Loan Amount, Loan Currency, Accrual Schedule ID, Accrual Schedule Efective Date, Spread Rate, Reserve Rate, Next rate Period date, Frequency, Lag days, Year Basis, Observation Shift, Primary Repayment Schedule Type, Next Due Date and Next Accrue To Date field values in Output file.");

Data.Store("NewLoanSetup1\_LoanNumber", LN1);

Data.Store("NewLoanSetup1\_LoanEffectiveDate", Loan1NewLoantabInfo[7]);

Data.Store("NewLoanSetup1\_CurrentSystemDate", Data.Fetch("ServicingLogin", "ApplicationProcessDate"));

Data.Store("NewLoanSetup1\_LoanMaturityDate", Loan1NewLoantabInfo[10]);

Data.Store("NewLoanSetup1\_LoanRateMaturityDate", Loan1NewLoantabInfo[9]);

Data.Store("NewLoanSetup1\_LoanAmount", Loan1NewLoantabInfo[5]);

Data.Store("NewLoanSetup1\_LoanCurrency", Loan1NewLoantabInfo[6]);

Data.Store("NewLoanSetup1\_AccrualScheduleID", Loan1AccId);

Data.Store("NewLoanSetup1\_AccrualScheduleEffectiveDate", LoanEffectiveDate);

Data.Store("NewLoanSetup1\_AccrualScheduleYearBasis", Loan1AccrualScheduleBasicInformationFieldValues[7]);

Data.Store("NewLoanSetup1\_AccrualScheduleSpreadRate", Loan1AccrualScheduleBasicInformationFieldValues[9]);

Data.Store("NewLoanSetup1\_AccrualScheduleReserveRate", Loan1AccrualScheduleBasicInformationFieldValues[10]);

Data.Store("NewLoanSetup1\_AccrualScheduleLagDays", Loan1AccrualScheduleBasicInformationFieldValues[40]);

Data.Store("NewLoanSetup1\_ObservationShift", Loan1AccrualScheduleBasicInformationFieldValues[42]);

Data.Store("NewLoanSetup1\_AccrualScheduleIndexRateChangeFrequency", Loan1AccrualScheduleBasicInformationFieldValues[45]);

Data.Store("NewLoanSetup1\_RepaymentScheduleType", Loan1RepaymentScheduleBasicInformationFieldValues[3]);

Data.Store("NewLoanSetup1\_RepaymentScheduleNextDueDate", NextDueDate);

Data.Store("NewLoanSetup1\_RepaymentScheduleNextAccrueToDate", NextDueDate);

Data.Store("NewLoanSetup1\_RepaymentScheduleNextDueDateFrequency", Loan1RepaymentScheduleBasicInformationFieldValues[11]);

Data.Store("NewLoanSetup1\_LoanInvoicePrintDate", LoanInvoicePrintDate);

}

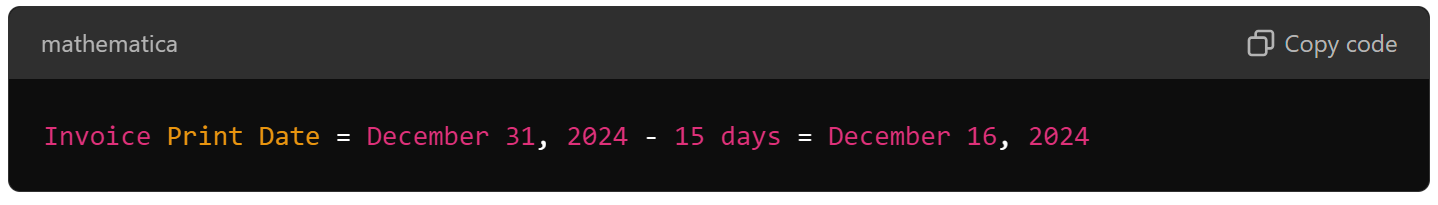
}

}

In this context, **invoicing** refers to the process of determining when the loan's invoice (billing) should be generated and sent to the customer. The **invoice print date** is the specific date when the invoice is printed or issued by the system.

The script calculates this invoice print date using **lead days**, which is a set number of days before the loan's next due date when the invoice should be generated. Here's how this works:

**Key Components:**

1. **Next Due Date:**
   * This is the date when the next payment (such as interest or principal) on the loan is due.
   * It is calculated earlier in the script based on the loan's effective date and repayment schedule. In this case, it might be the effective date of the loan plus a couple of months (e.g., "Loan Effective Date + 2 Months").
2. **Lead Days:**
   * Lead days represent the number of days before the **Next Due Date** that the invoice needs to be printed or prepared. For example, if the lead days are 15, the invoice will be printed 15 days before the due date.
3. **Invoice Print Date Calculation:**
   * The script calculates the invoice print date by subtracting the lead days from the next due date. The formula might look something like this:
   * This ensures that the invoice is generated well in advance of the due date, giving the customer time to review the invoice and make payments.
4. **Code Reference:**
   * In the script, this calculation is done using the method ACBSUtility.ACBSUtilityLibrary.CalculateNewDateWithLagDays(). This function likely takes the next due date and subtracts the lead days (which are stored in the repayment schedule information) to determine the exact invoice print date.

**Example:**

If the loan’s **Next Due Date** is December 31, 2024, and the **Lead Days** are 15, the system would calculate the **Invoice Print Date** as follows:

mathematica

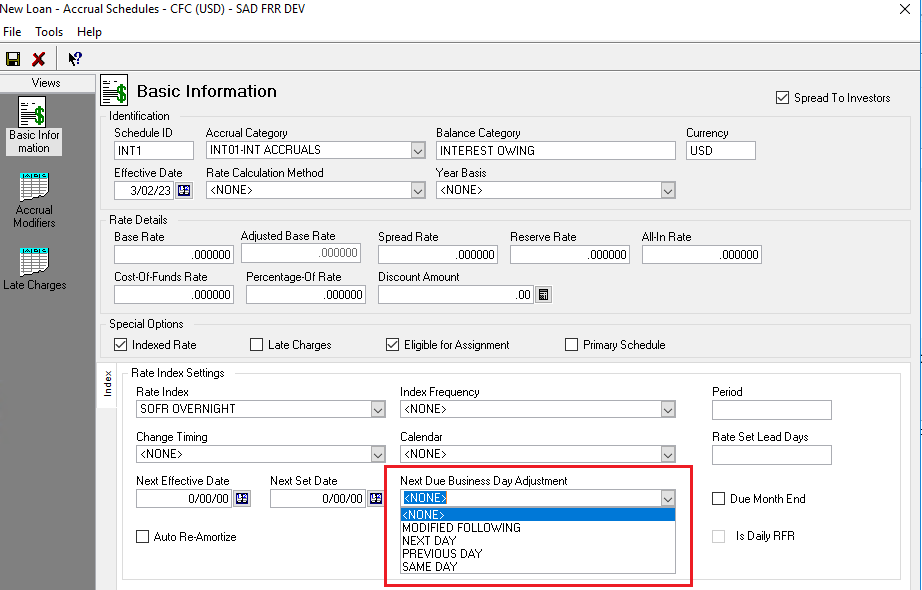
Copy code

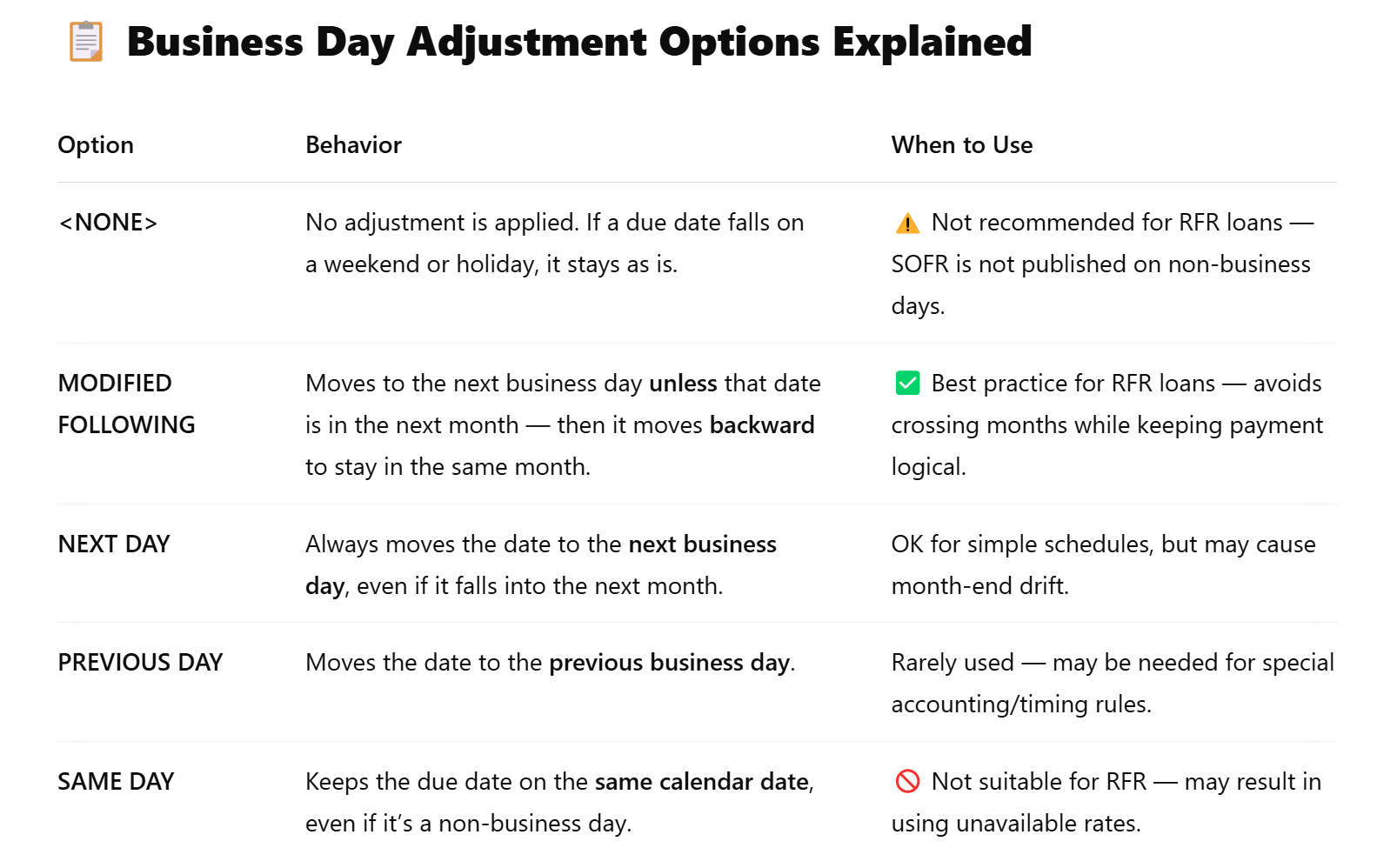
Invoice Print Date = December 31, 2024 - 15 days = December 16, 2024

The script then stores this calculated invoice print date for further use, such as generating or scheduling the actual invoice within the loan servicing system.

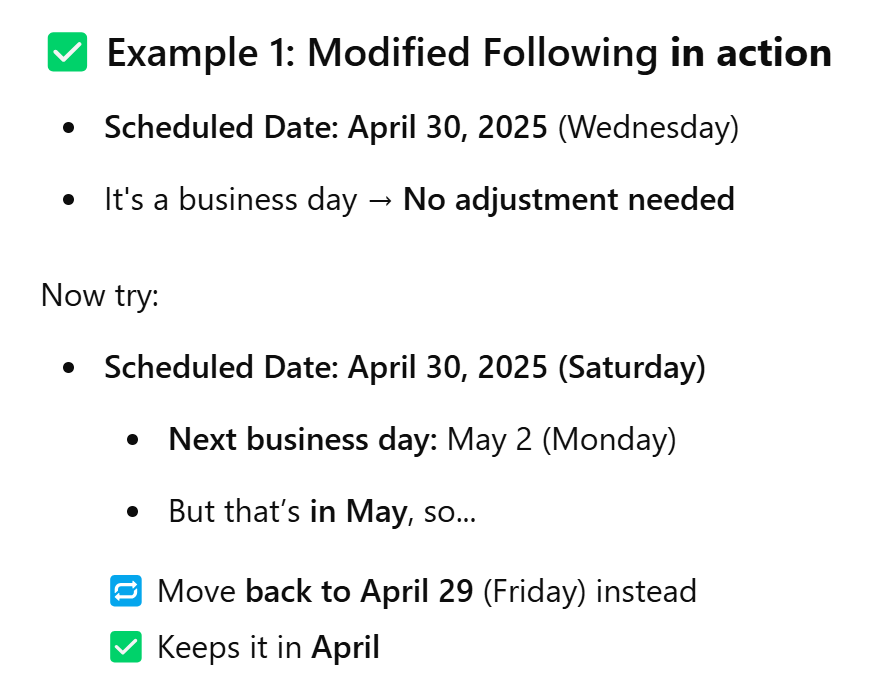
**Purpose:**

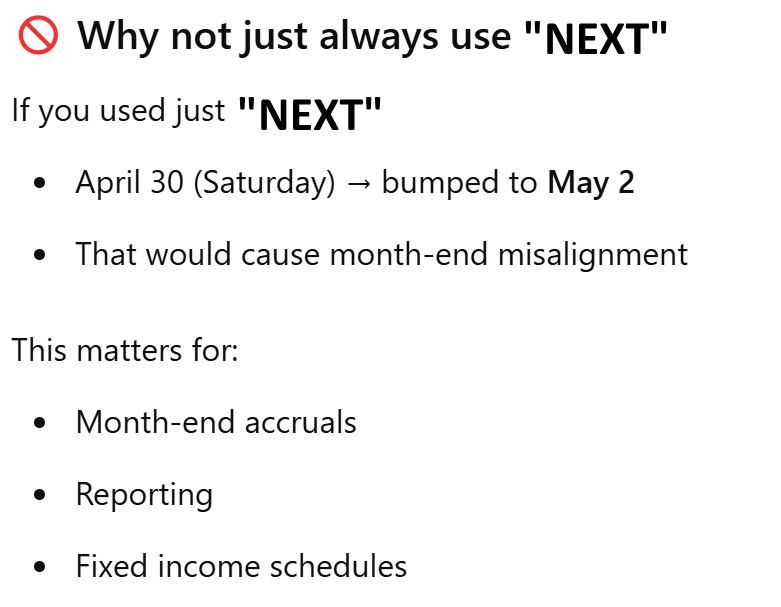
This process ensures that invoices are generated in a timely manner, allowing borrowers to receive their billing information before payments are due. It helps automate the billing process and ensures accuracy in scheduling.

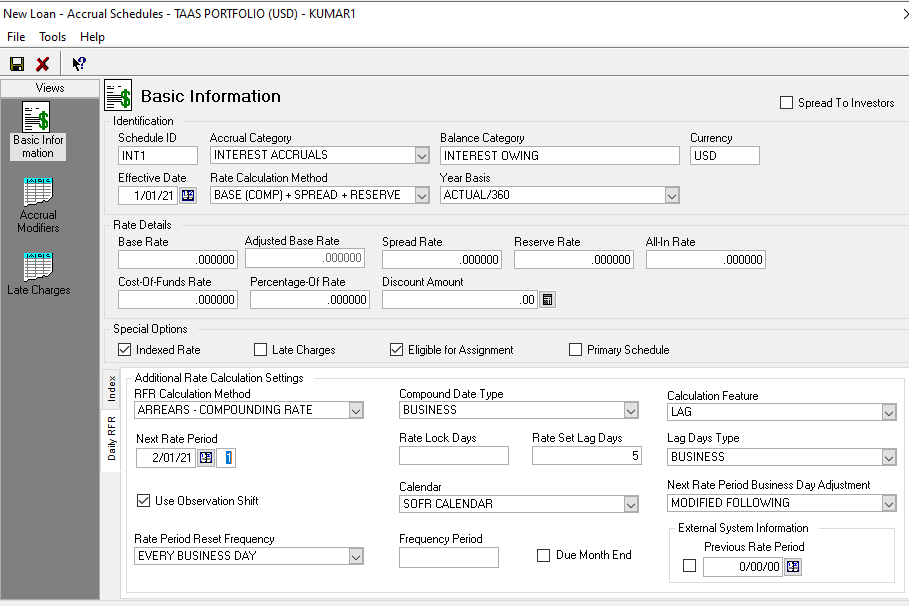


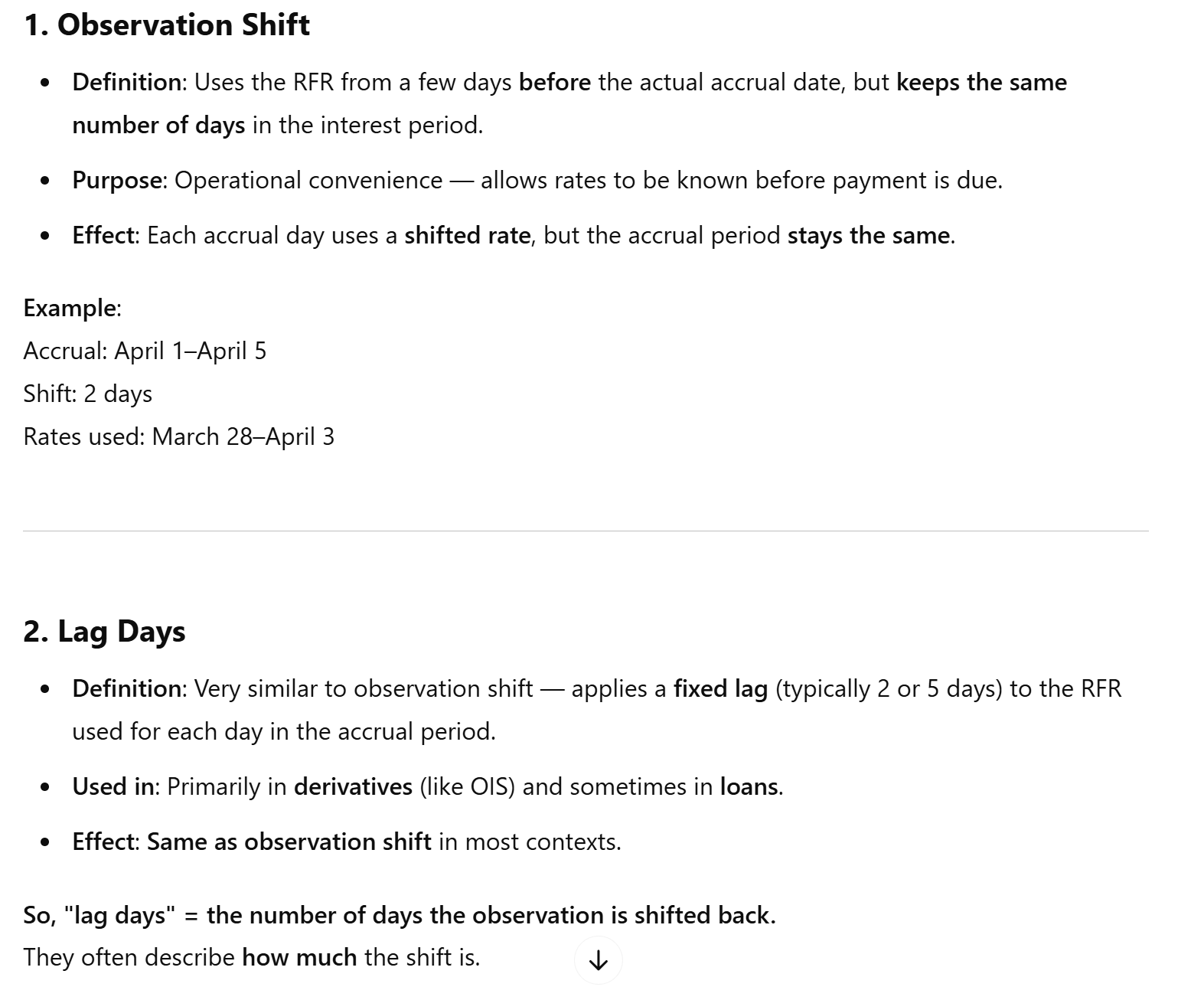


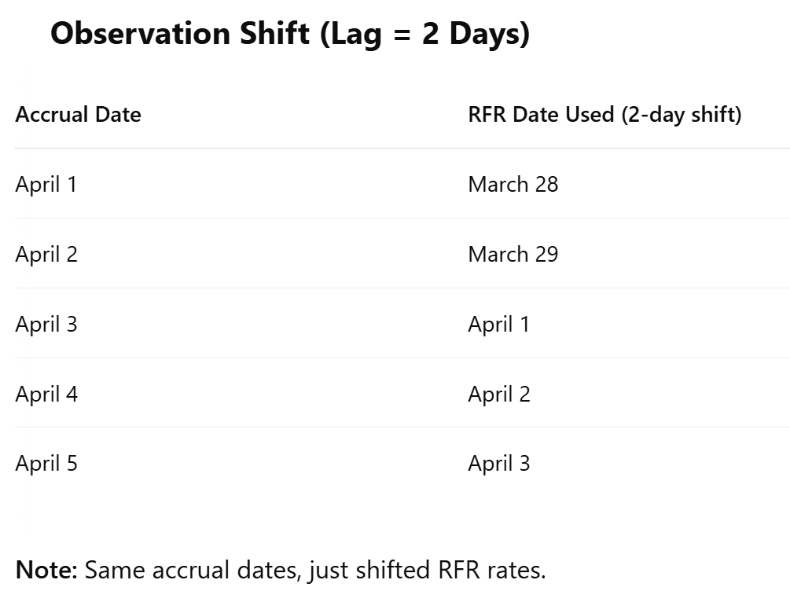


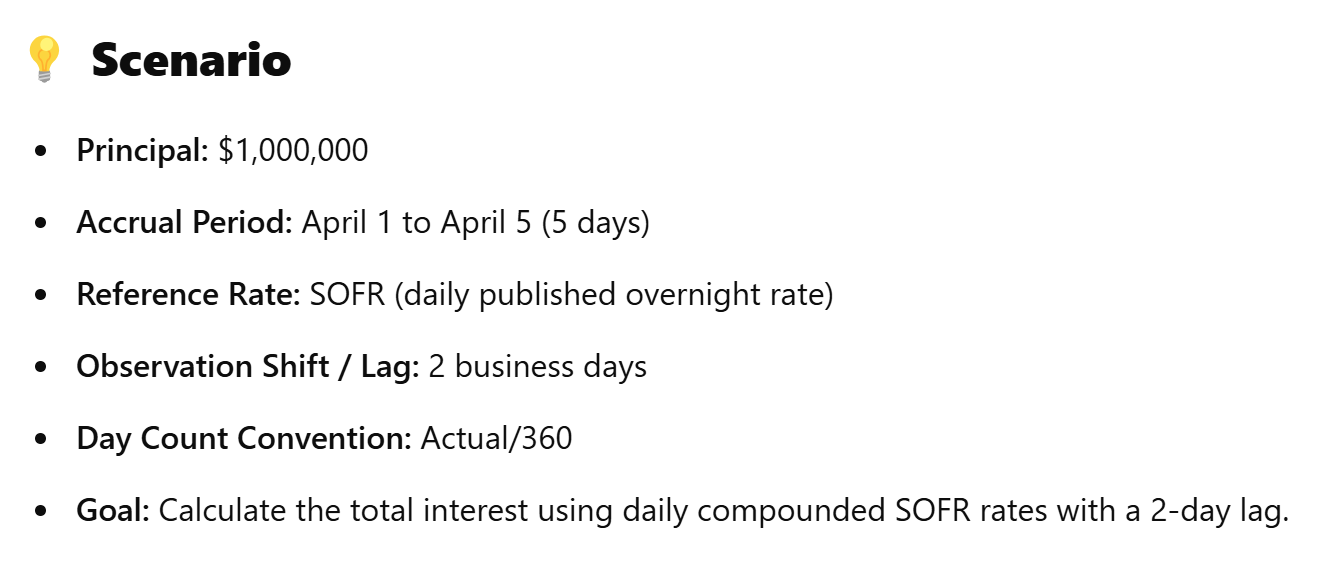


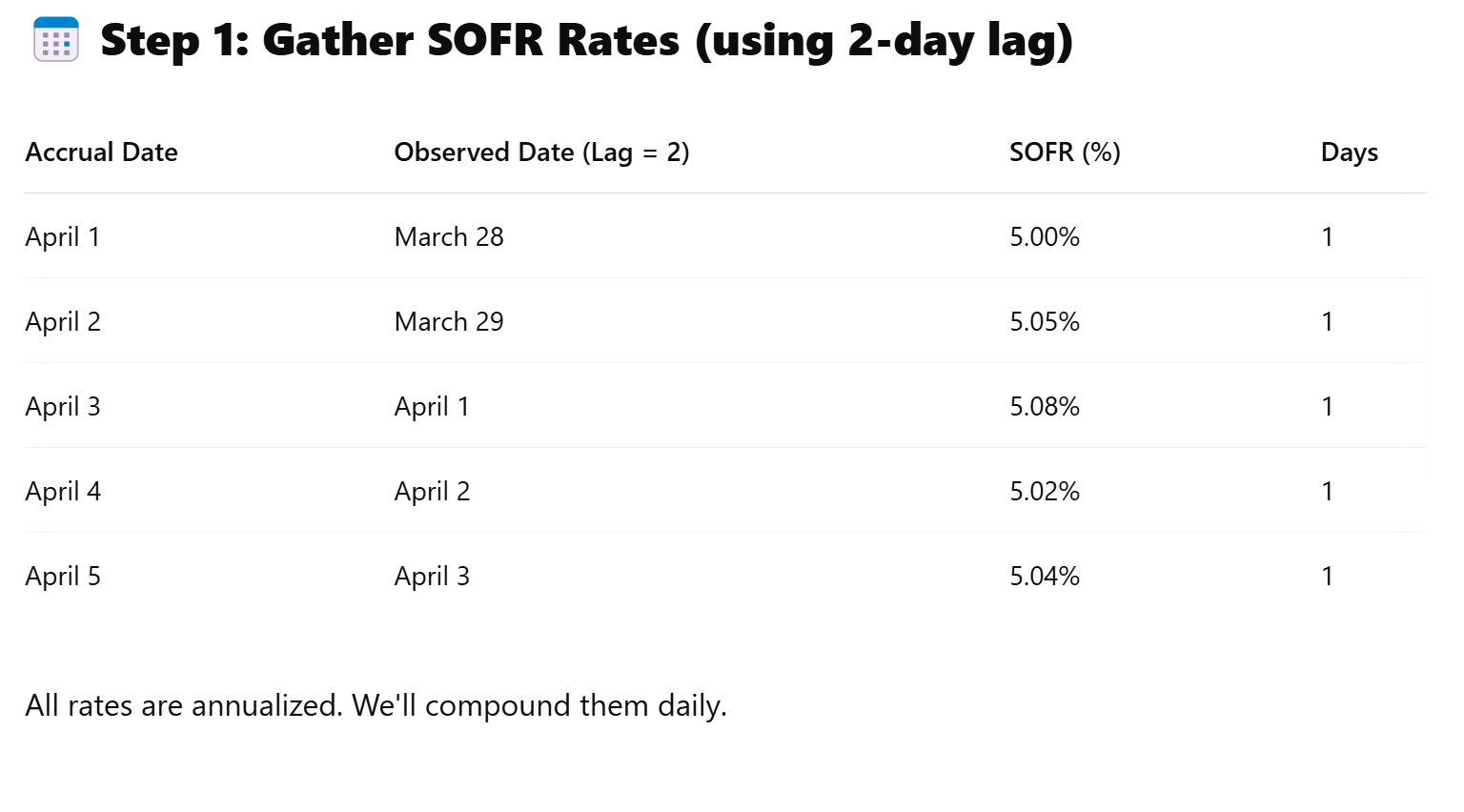


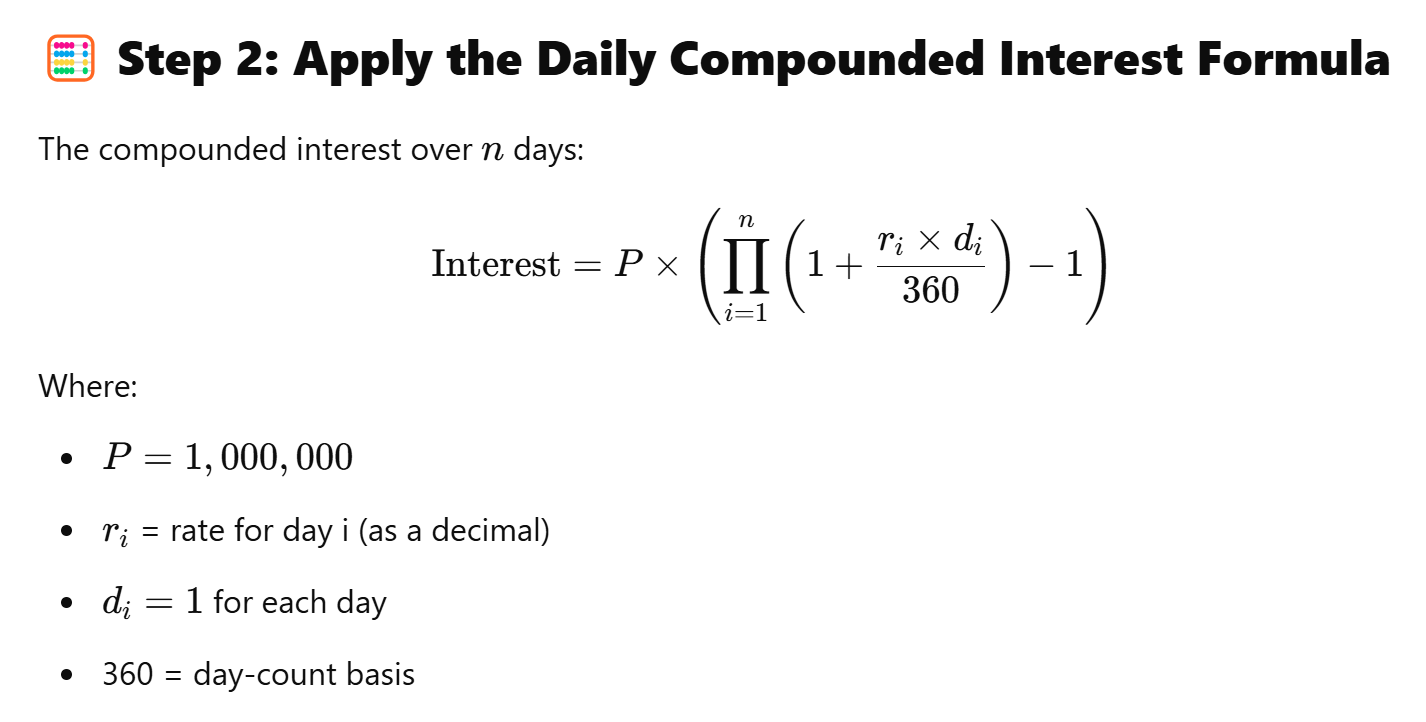


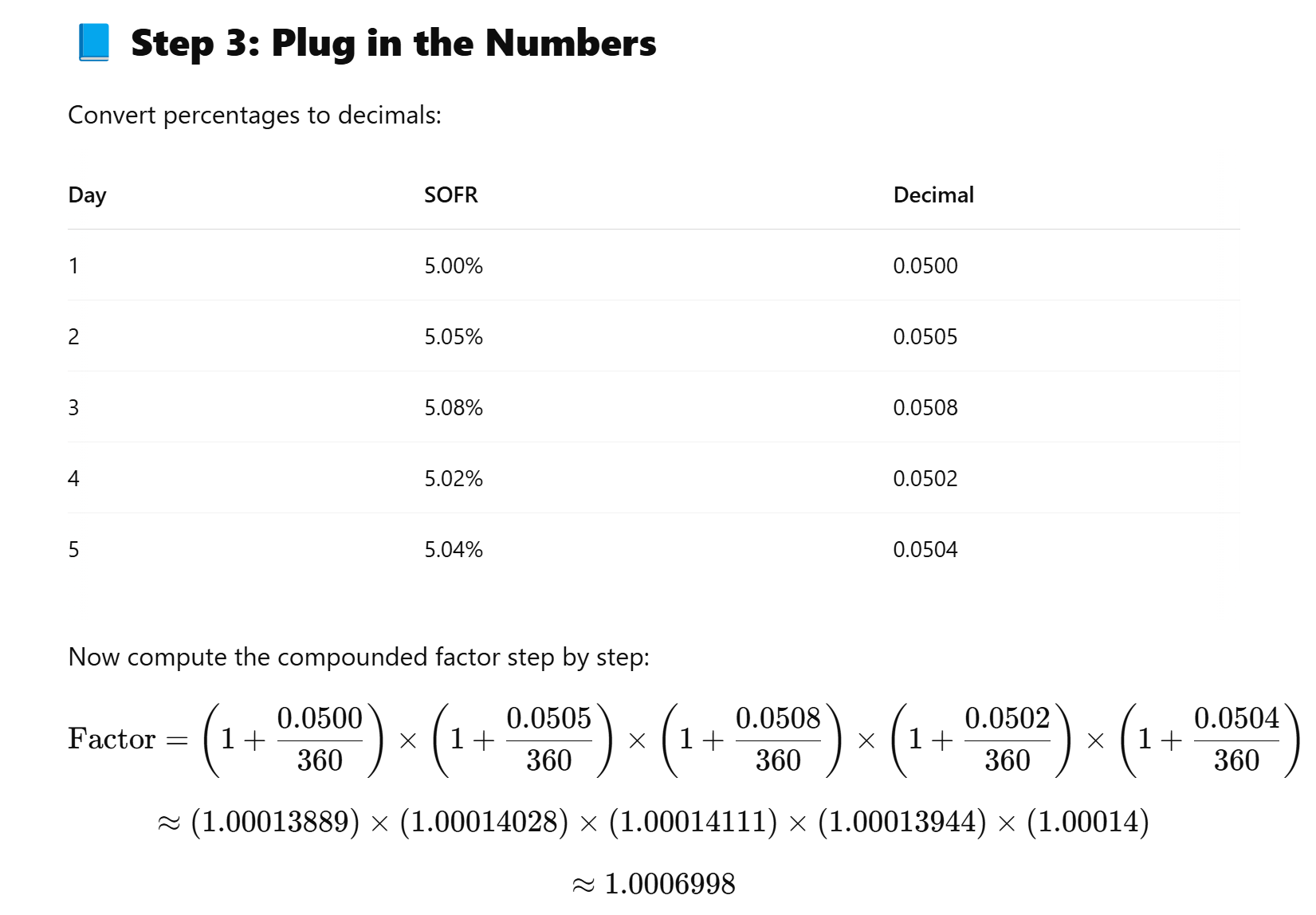


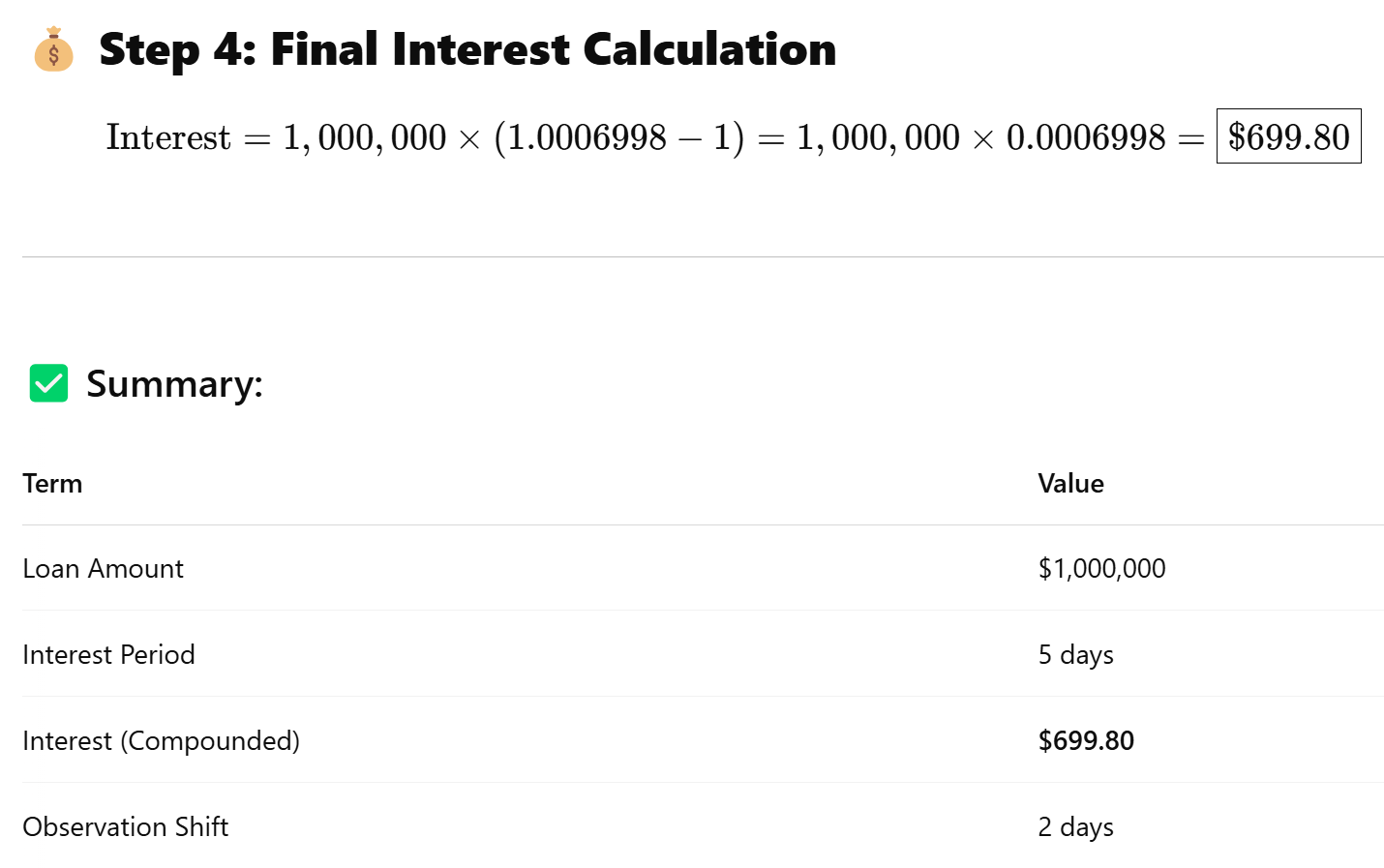


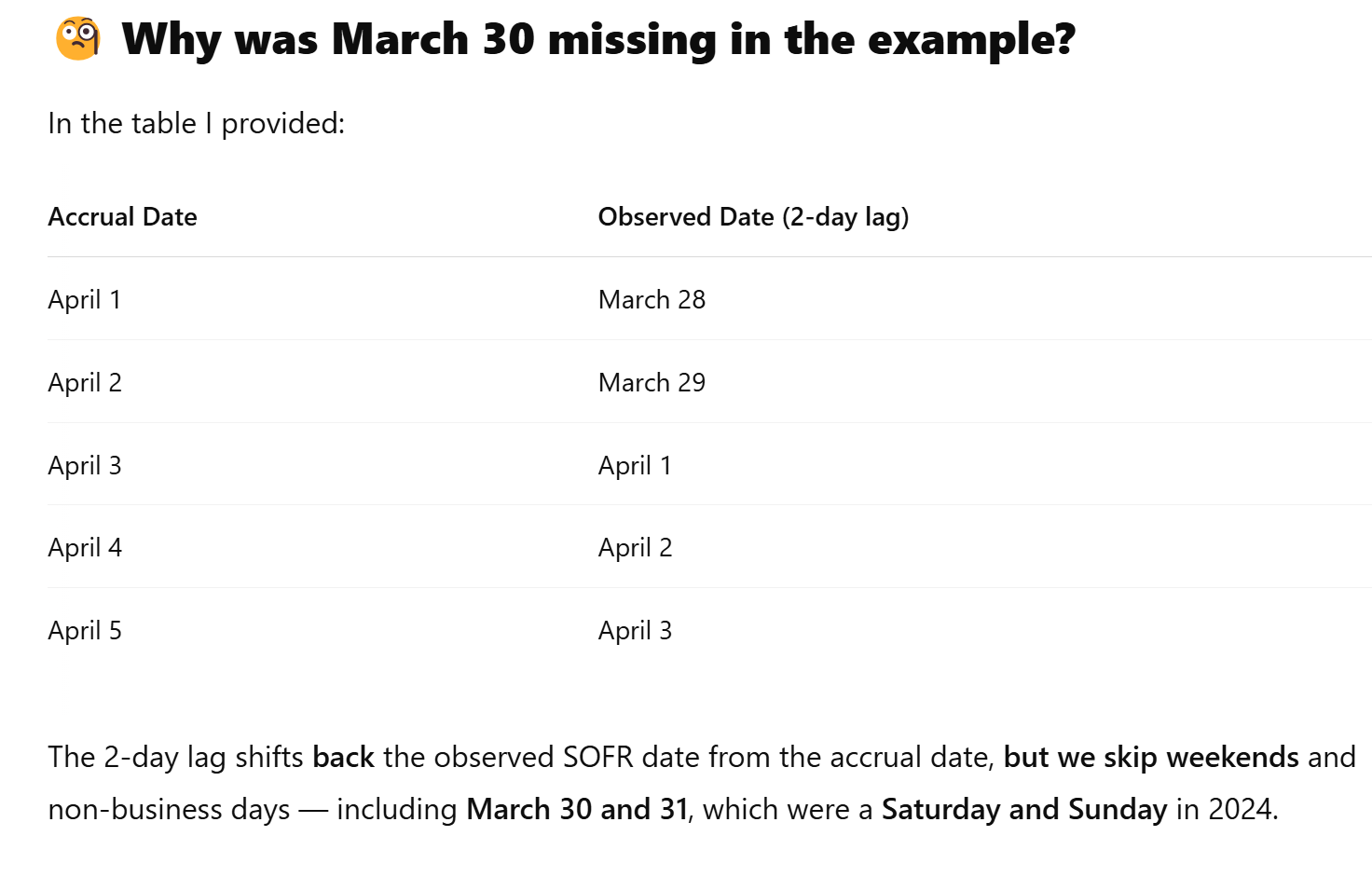


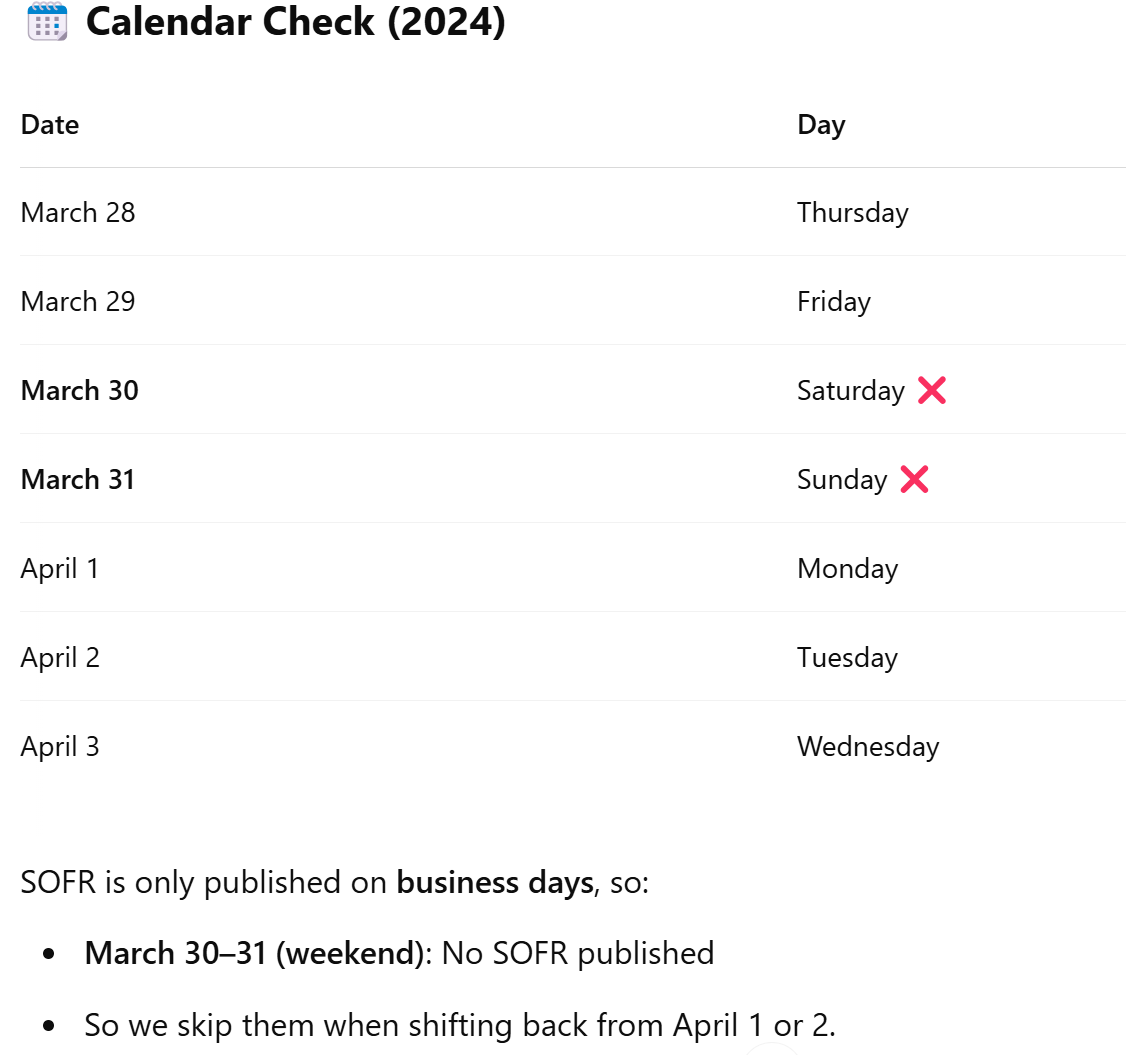


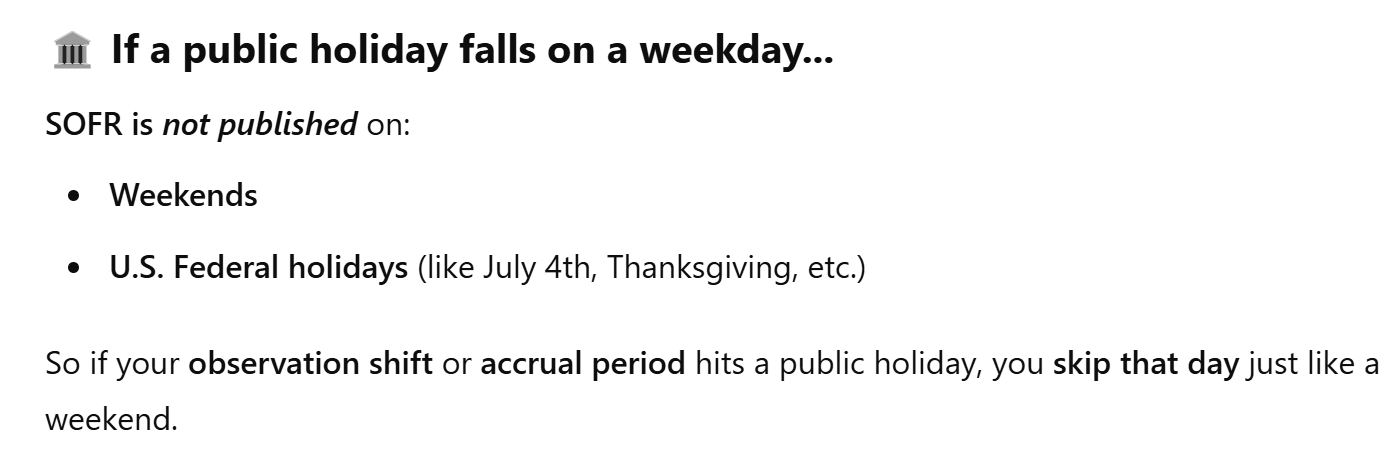






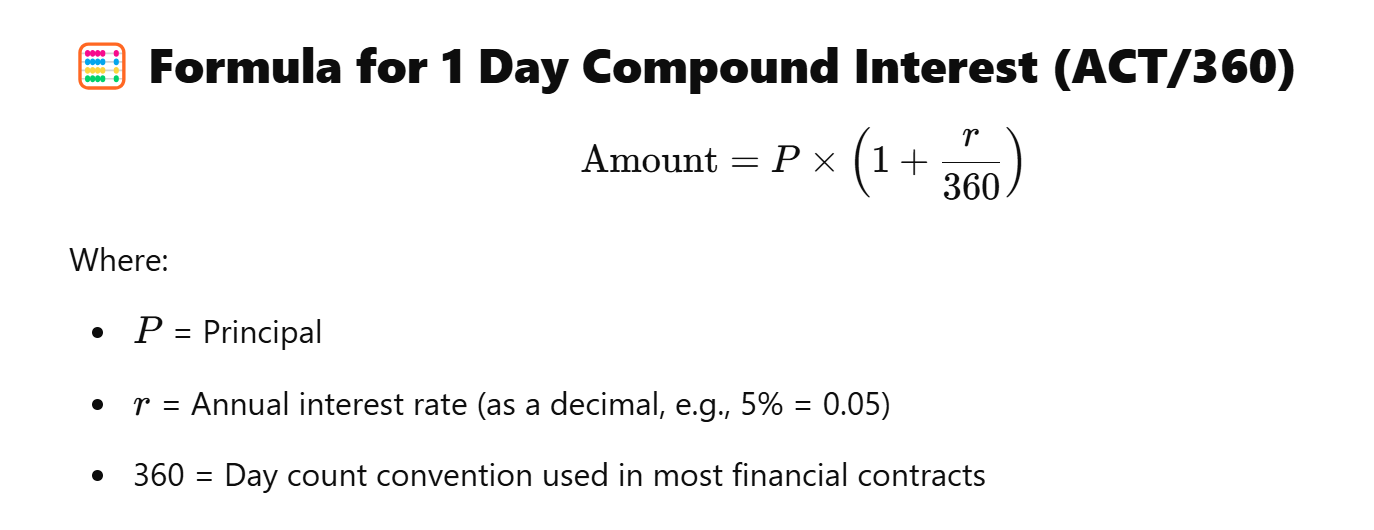


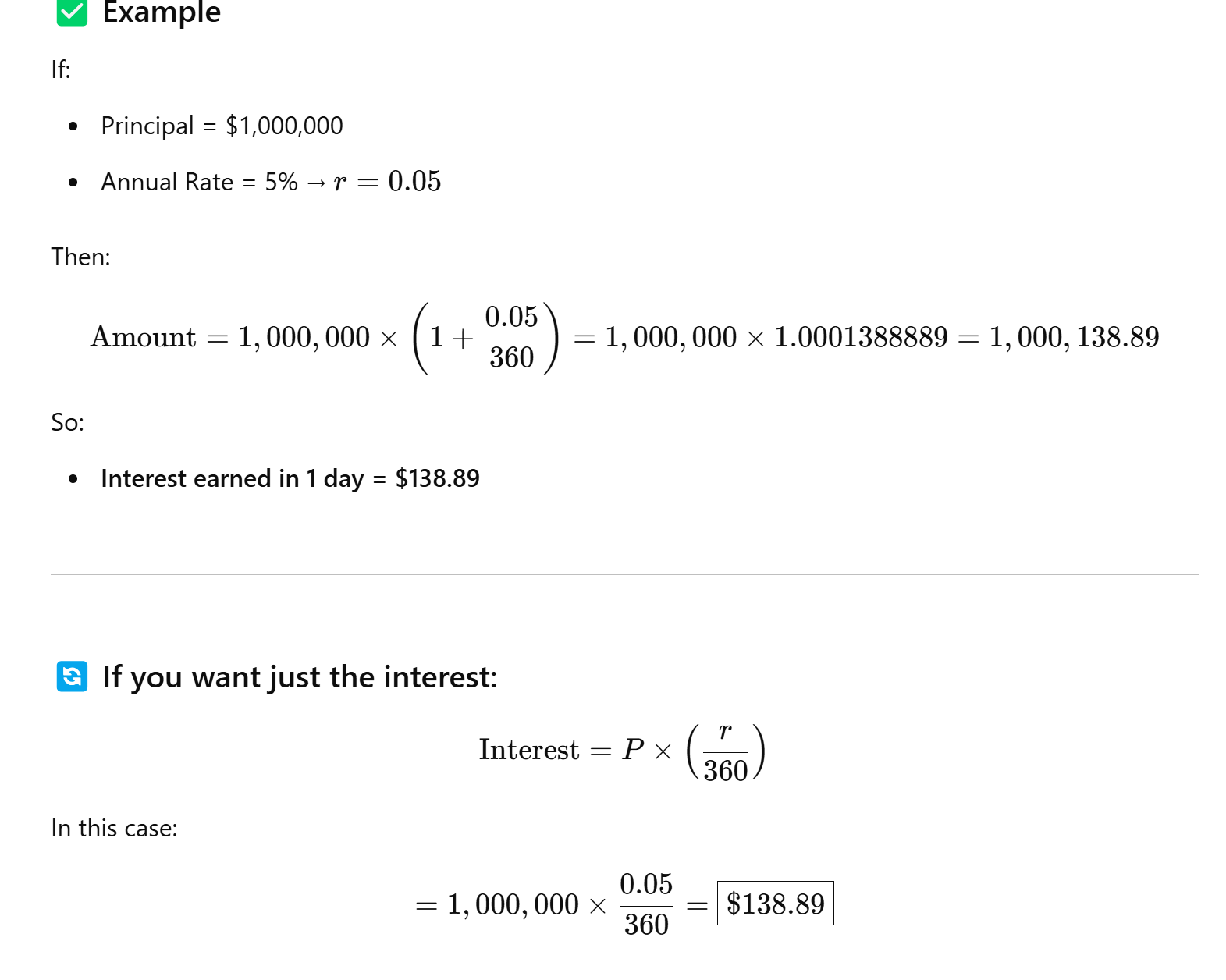


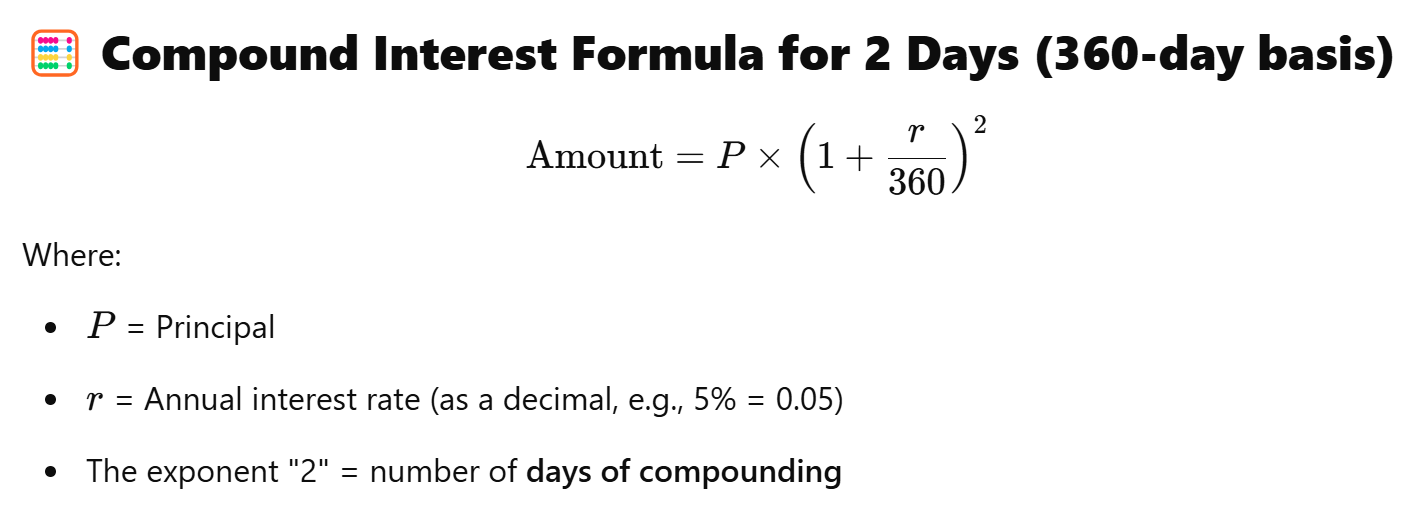


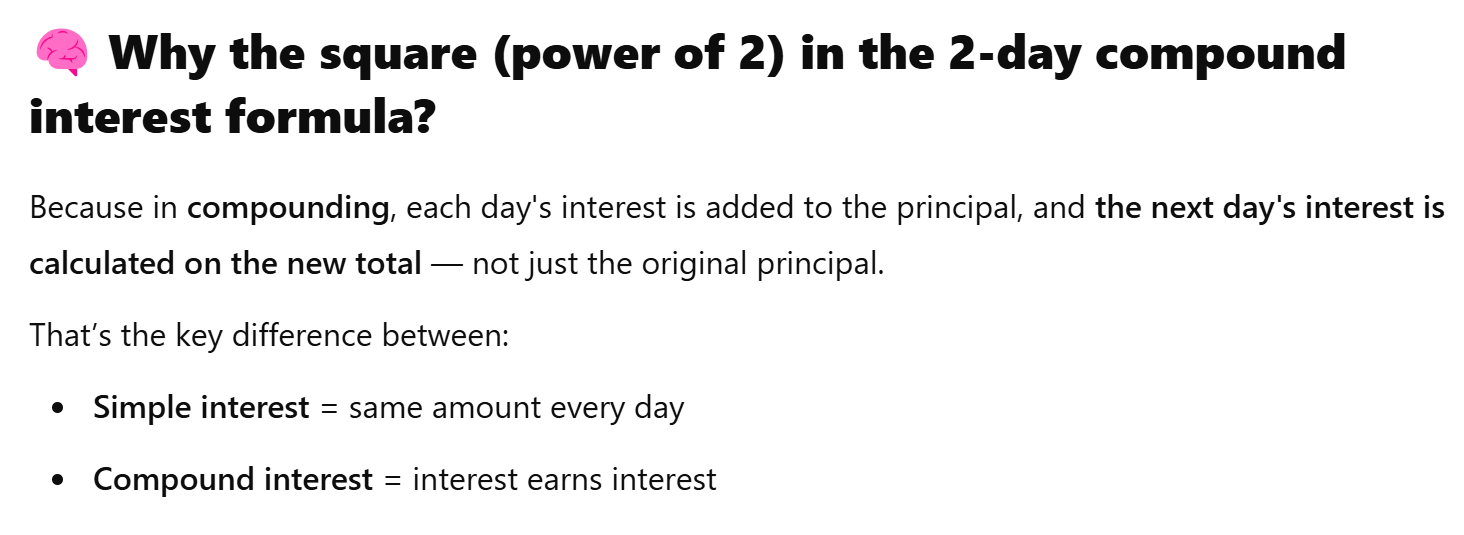


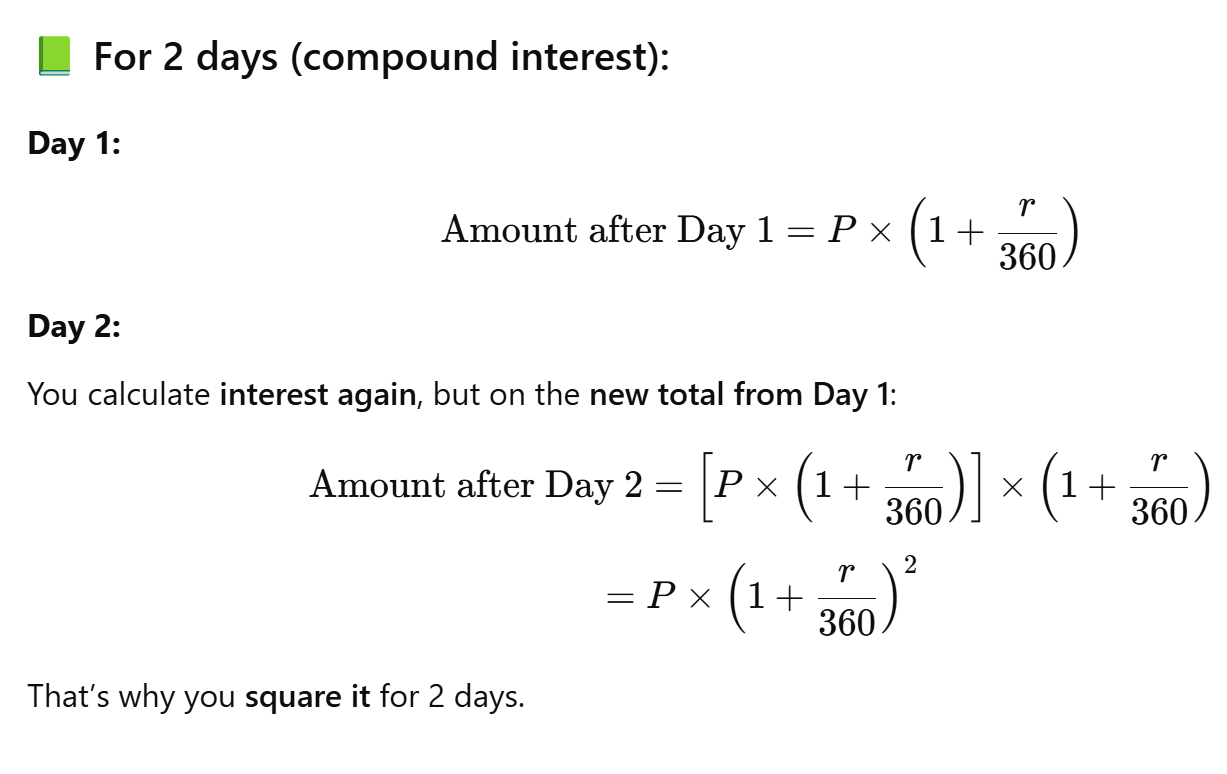
**Compound Interest formula derivation:**

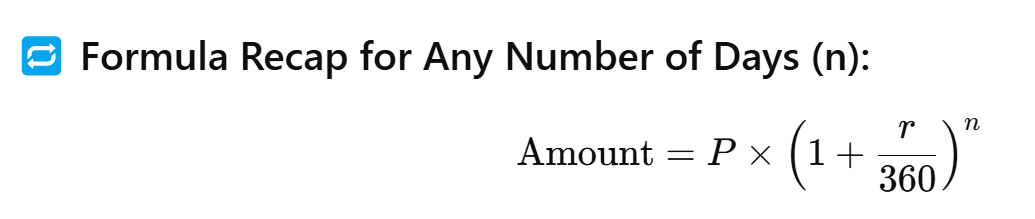
****

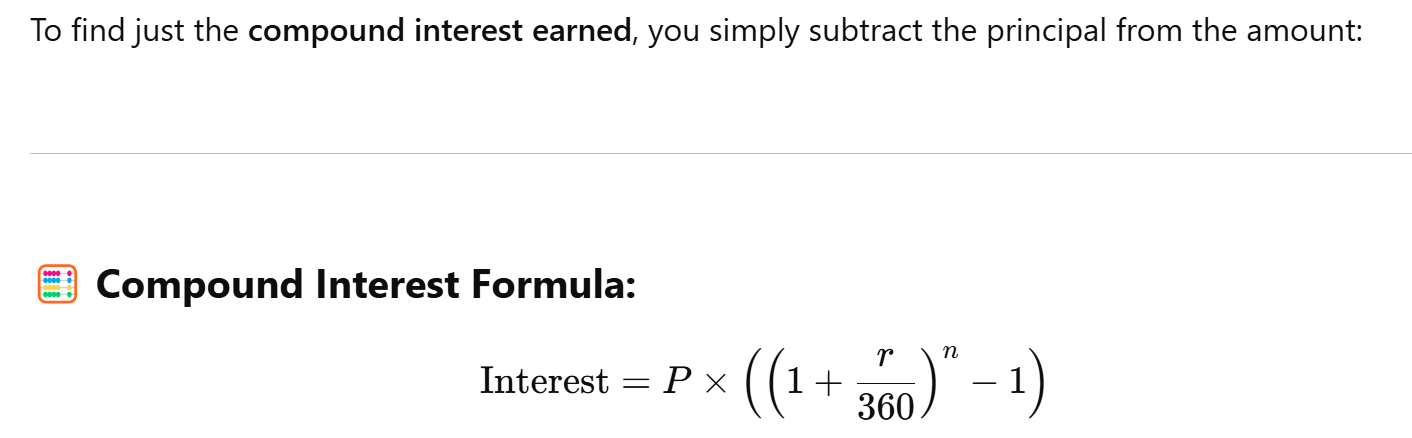
****

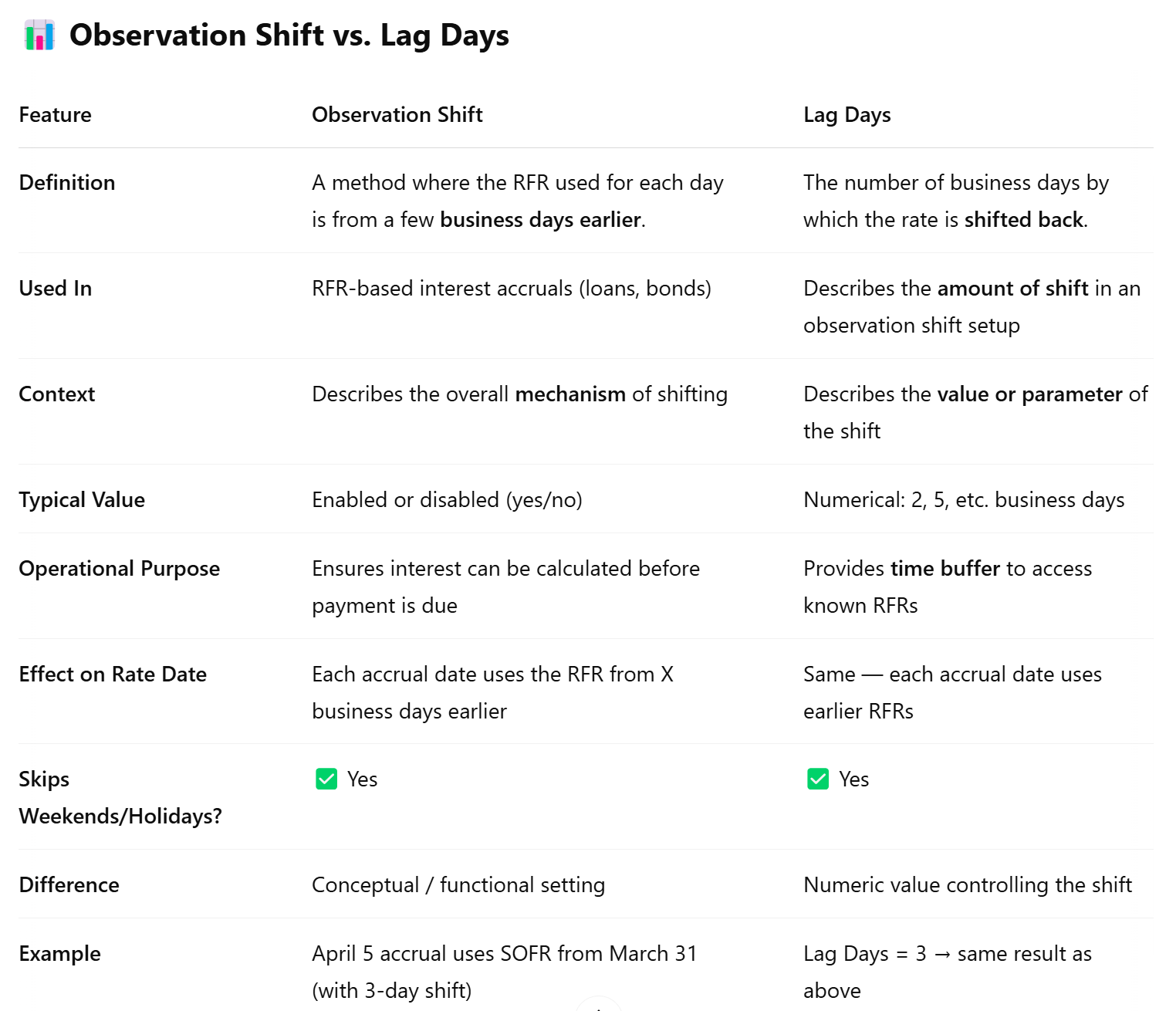
****

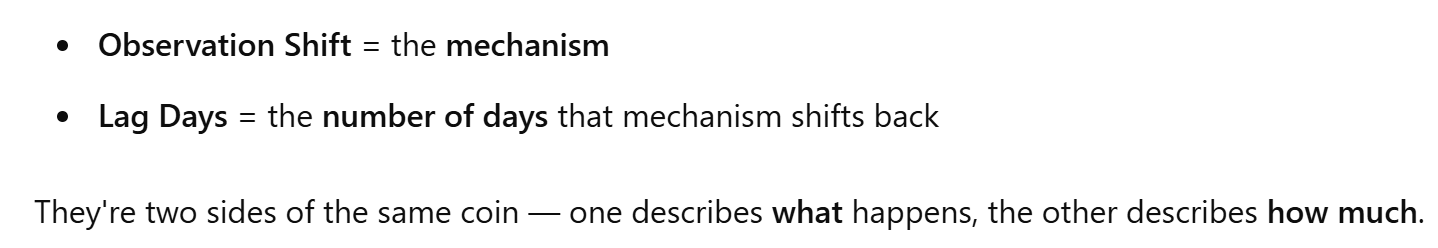
****

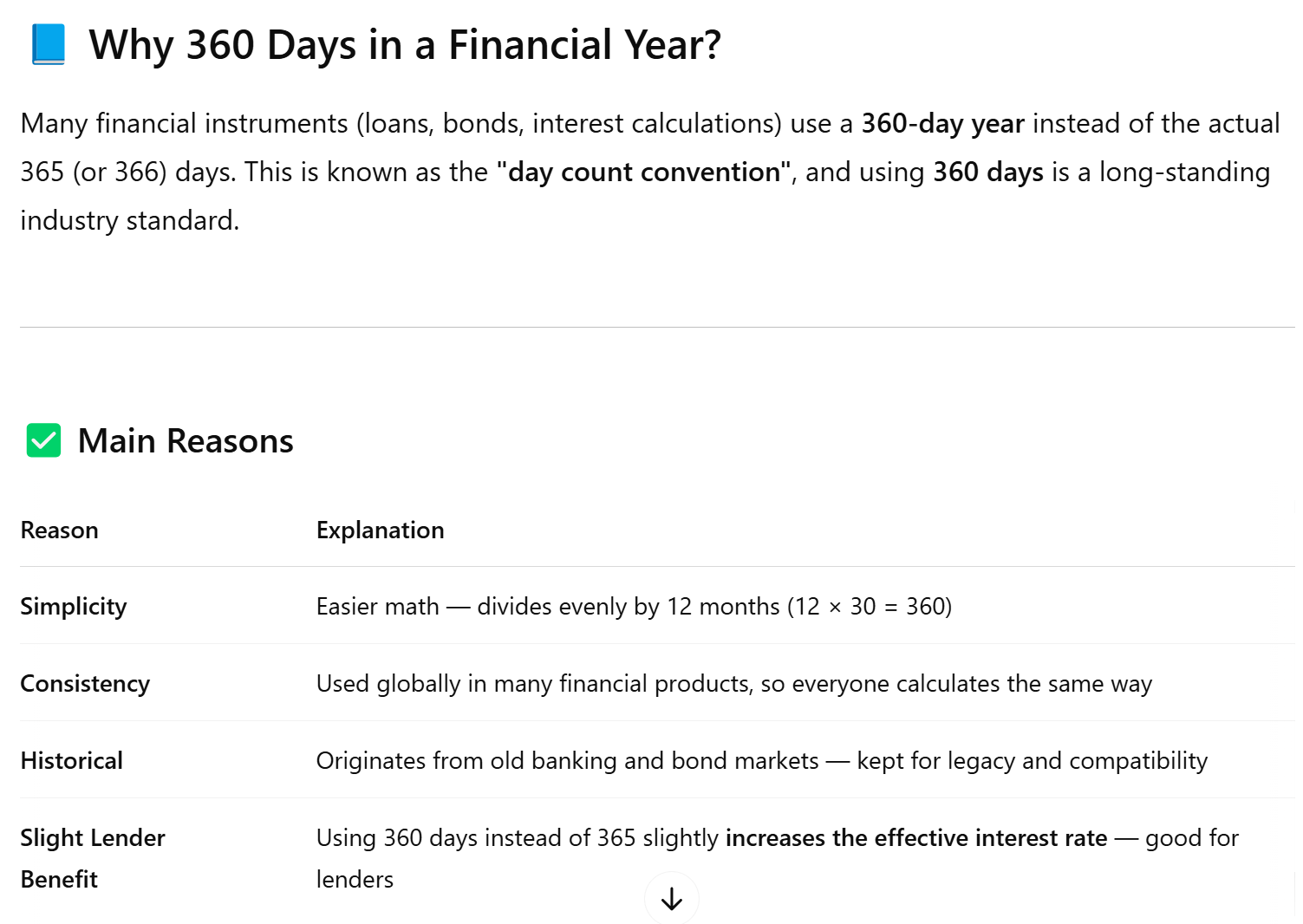
****

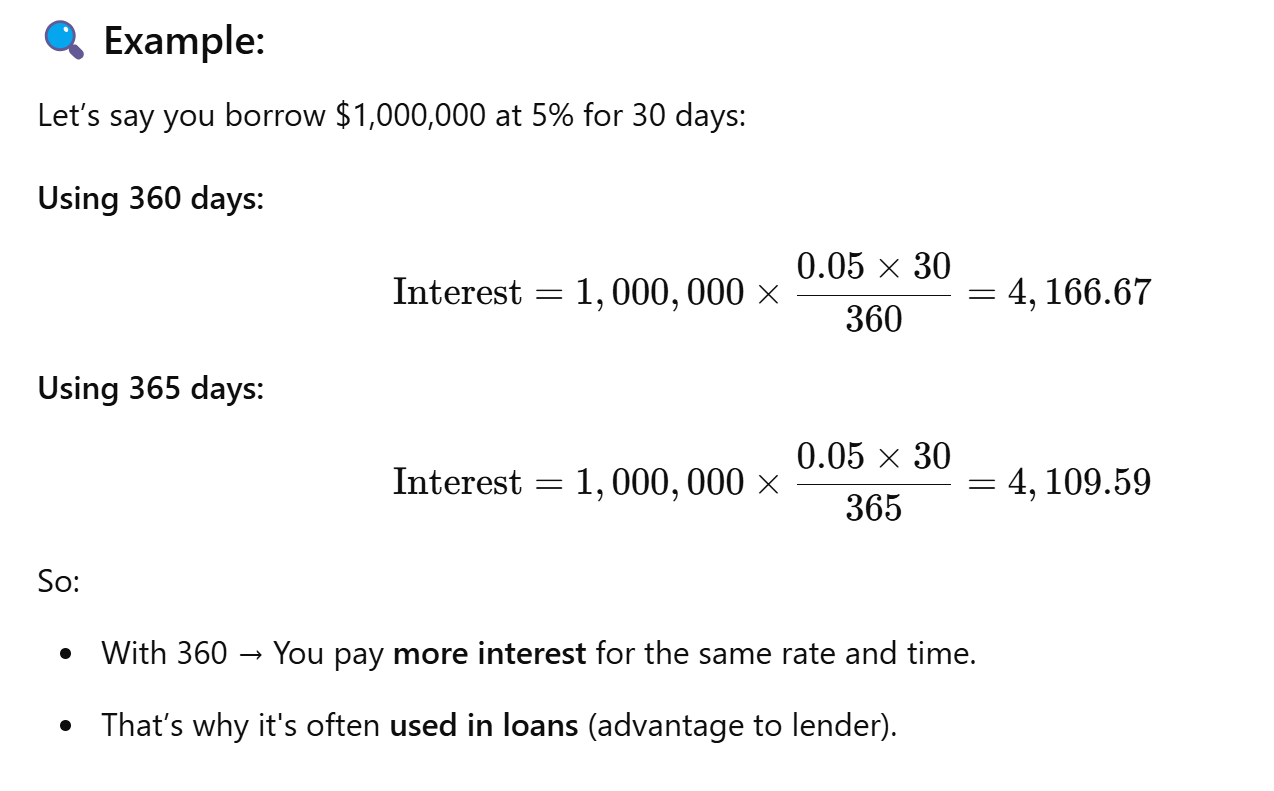
****

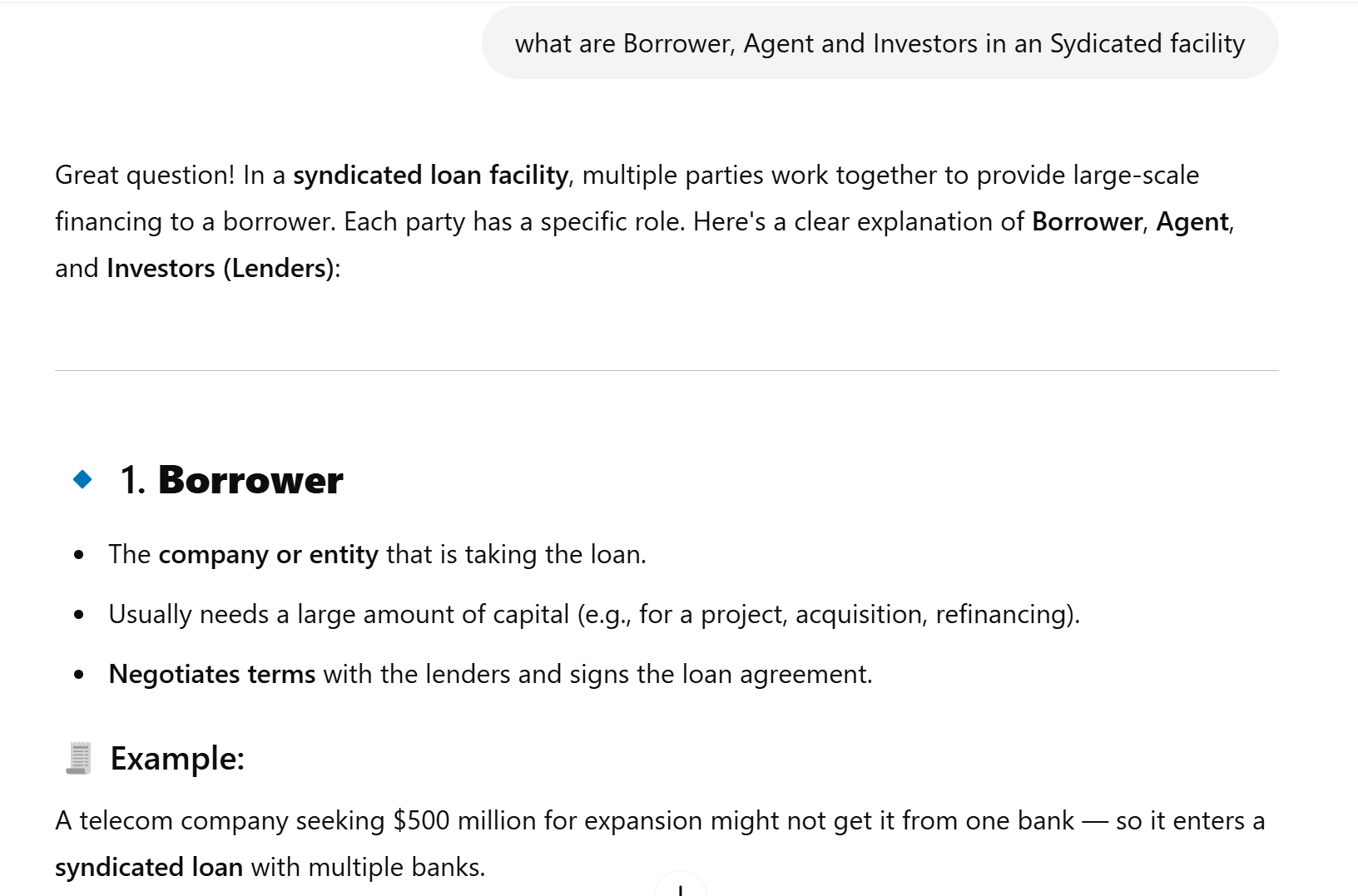
****

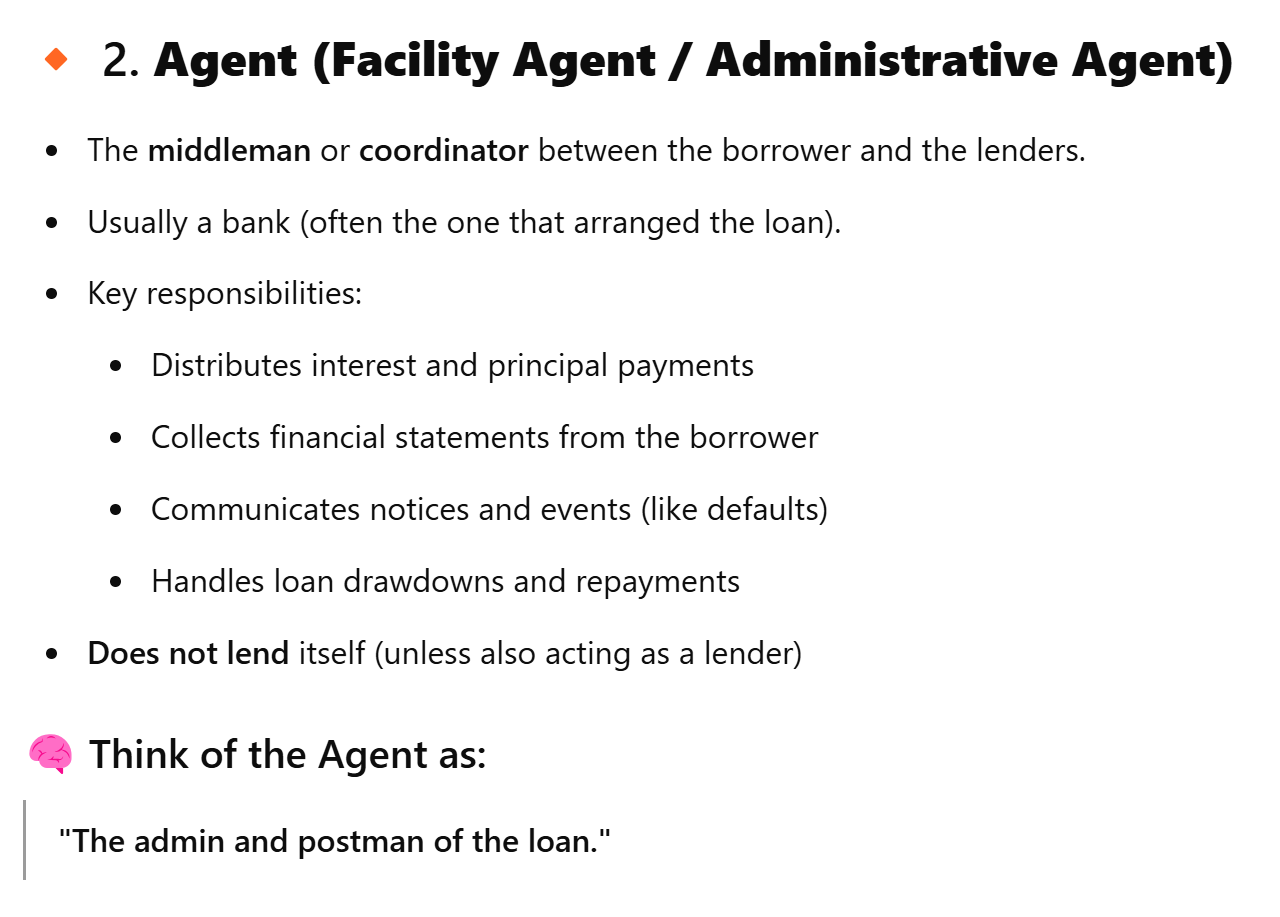


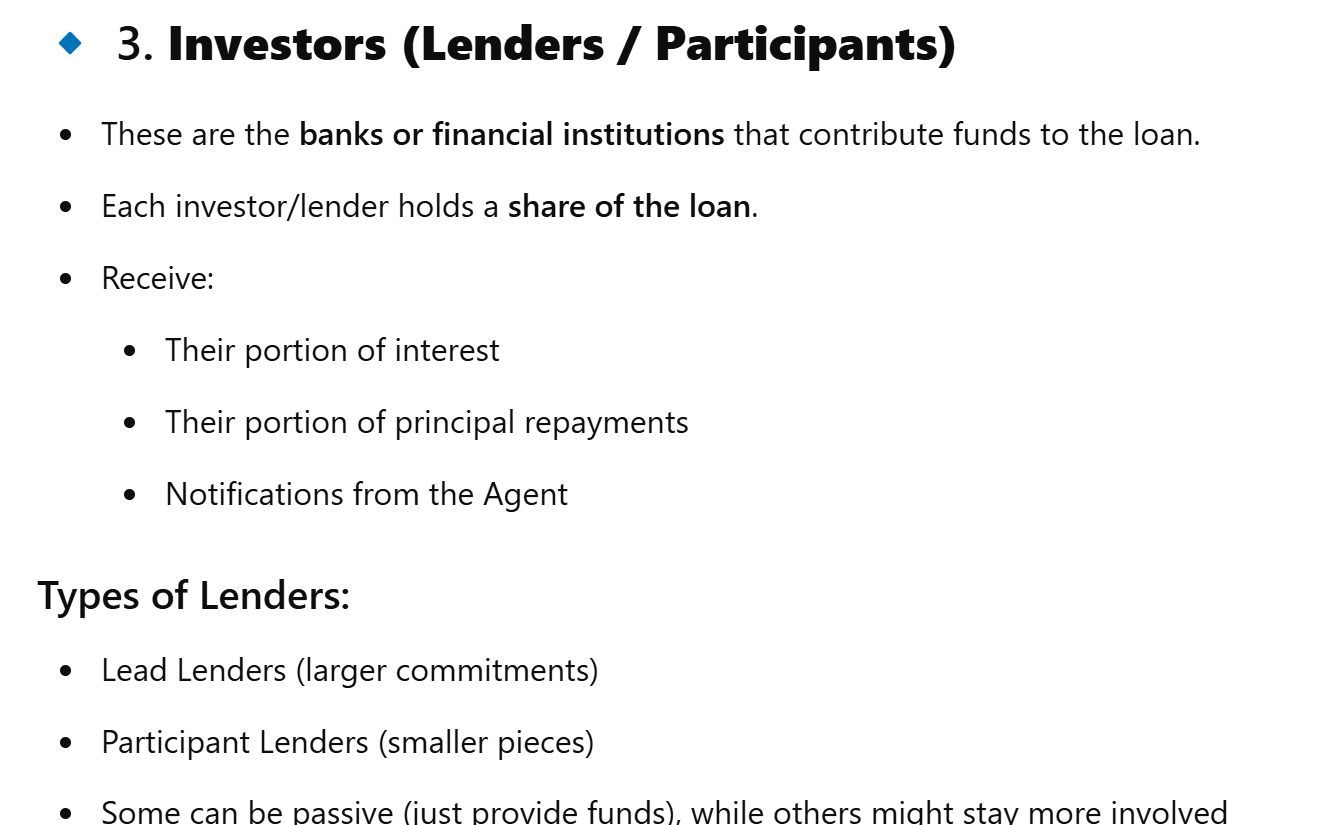


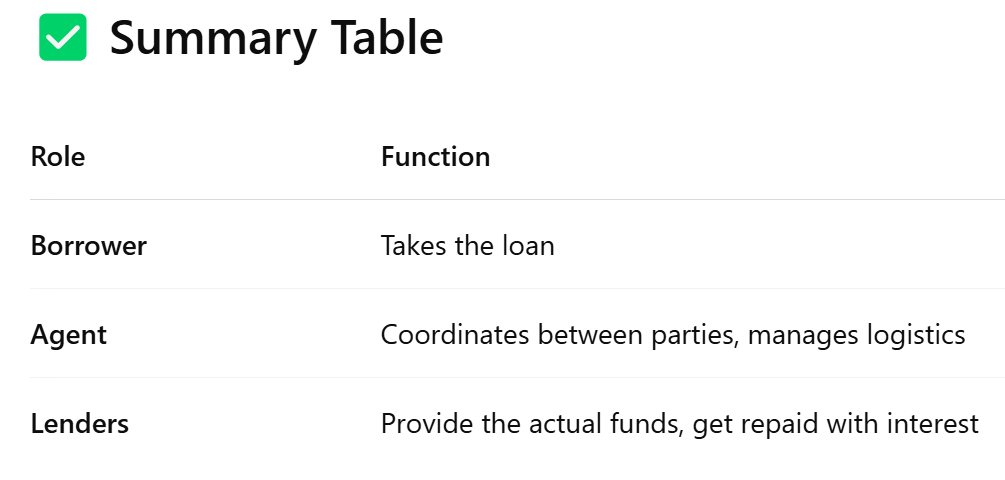


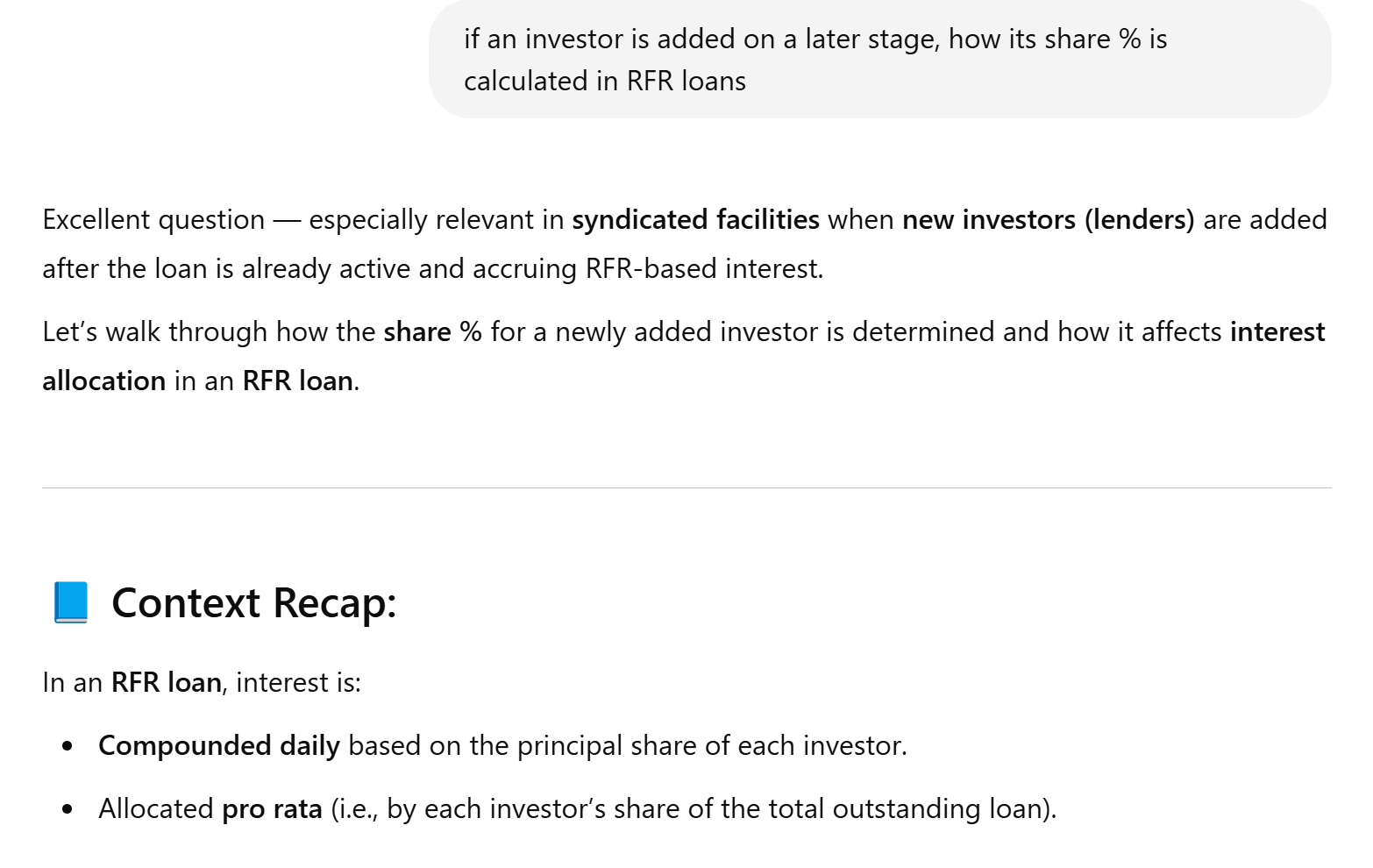




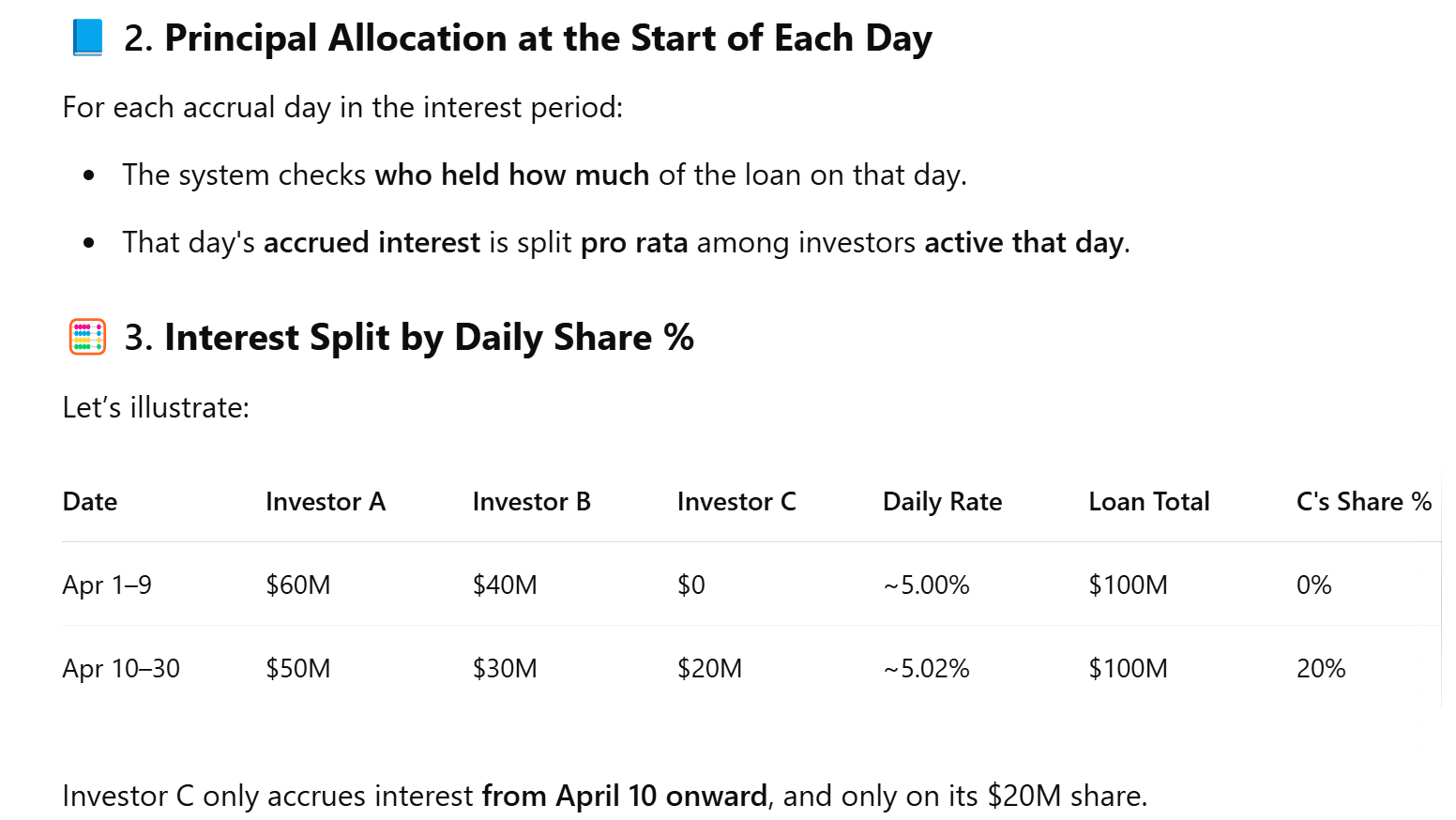


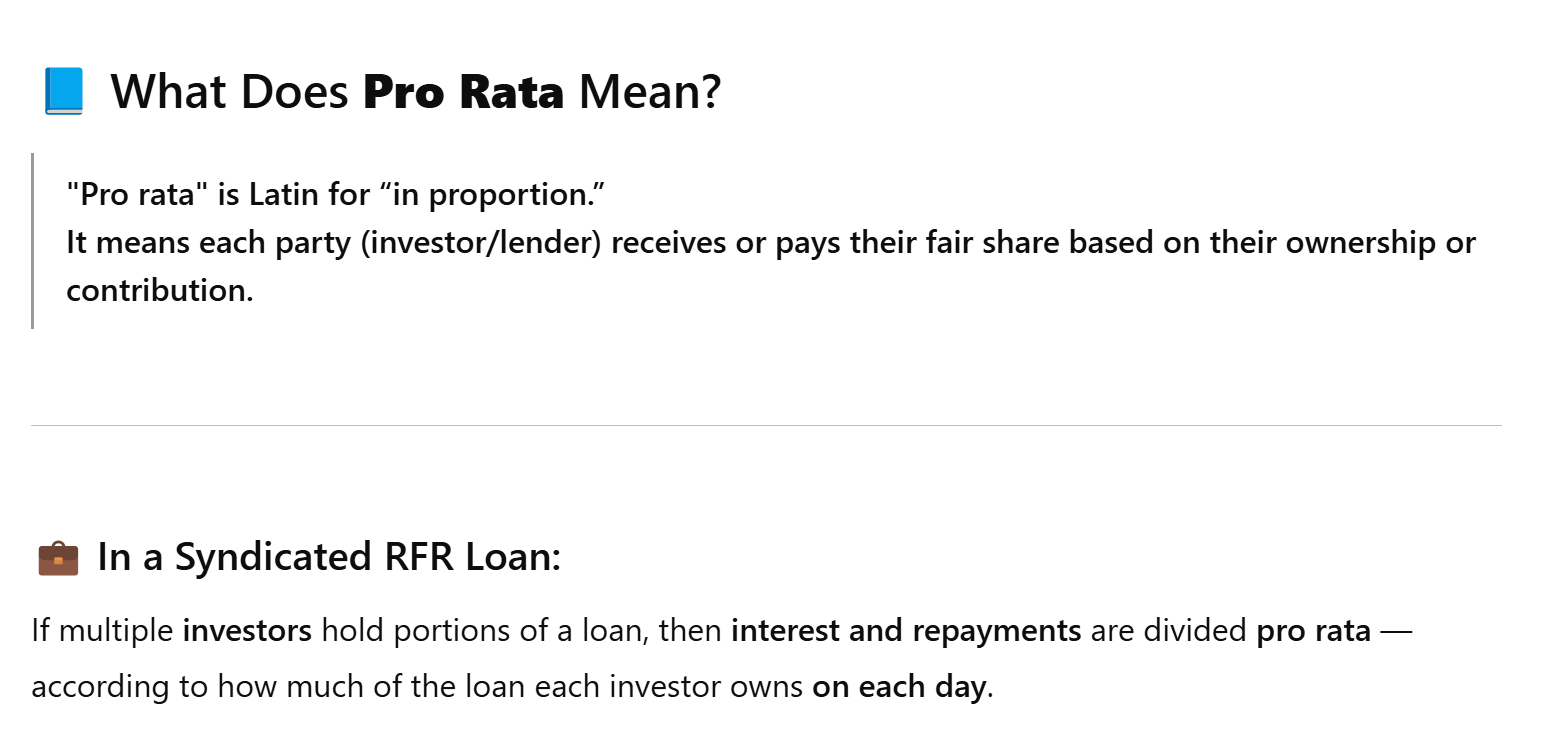








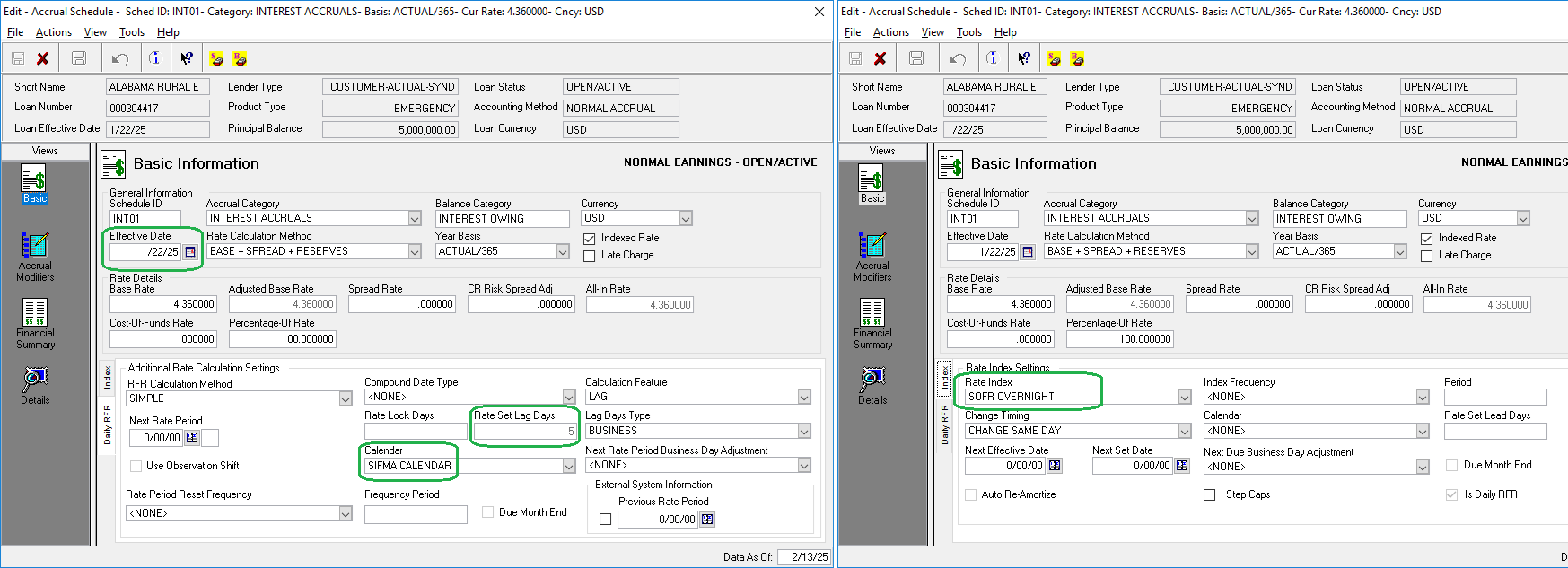


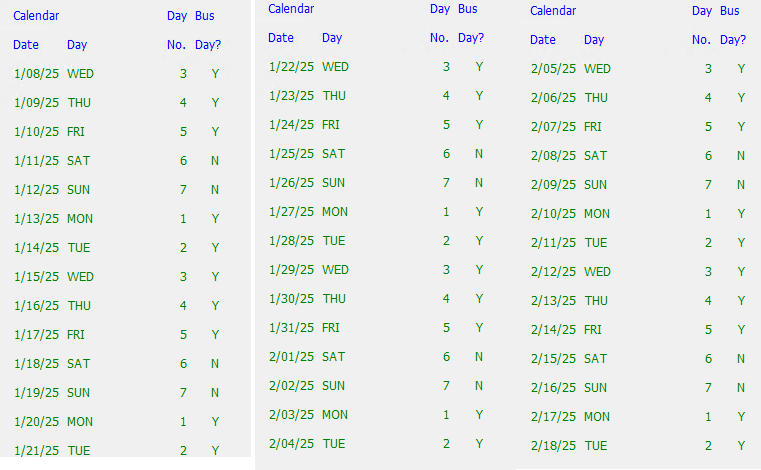




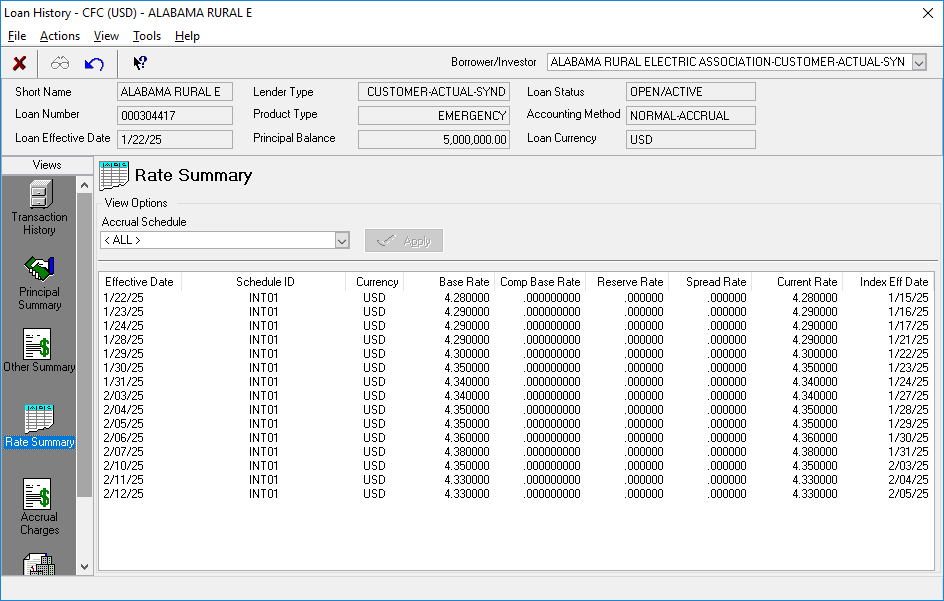


**Index Rate Summary:**









Let us consider 1st row, i.e. 1/22/25

**Meaning of Lag days:** Lag days are always backward looking, and we don’t directly substract lag days from eff date, rather we substract Business days equal to Lag day.

E.g. it is not like: 22nd Jan – 5 days = 17th Jan then we consider if 17th is business day or not

Rather, it is done on individual day basis as explained below:

Provided 5 days lag in Accrual Schedule:

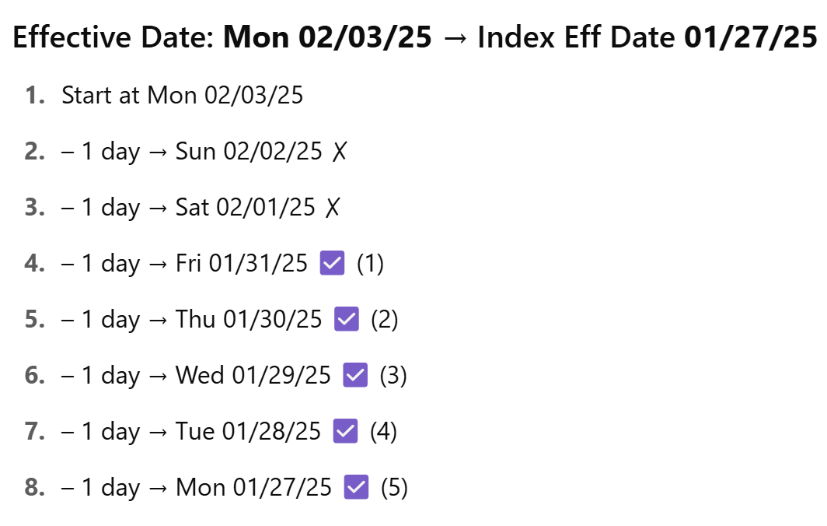
1. **Start at Wed 01/22/25**
2. **– 1 day → Tue 01/21/25 ☑ business day (count = 1)**
3. **– 1 day → Mon 01/20/25 ☑ business day (count = 2)**
4. **– 1 day → Sun 01/19/25 ✖ weekend (skip)**
5. **– 1 day → Sat 01/18/25 ✖ weekend (skip)**
6. **– 1 day → Fri 01/17/25 ☑ business day (count = 3)**
7. **– 1 day → Thu 01/16/25 ☑ business day (count = 4)**
8. **– 1 day → Wed 01/15/25 ☑ business day (count = 5)**

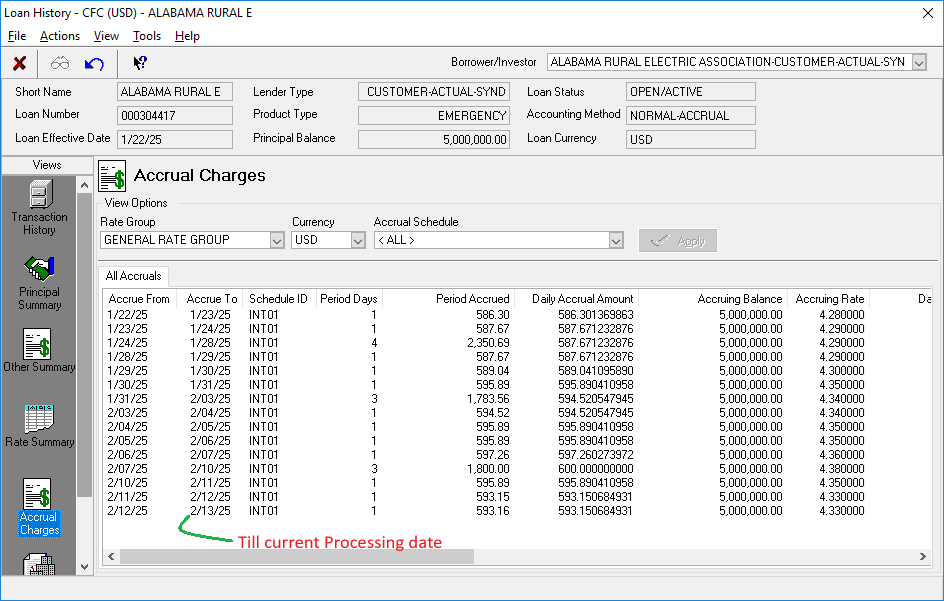
**→ Index Eff Date = 01/15/25**

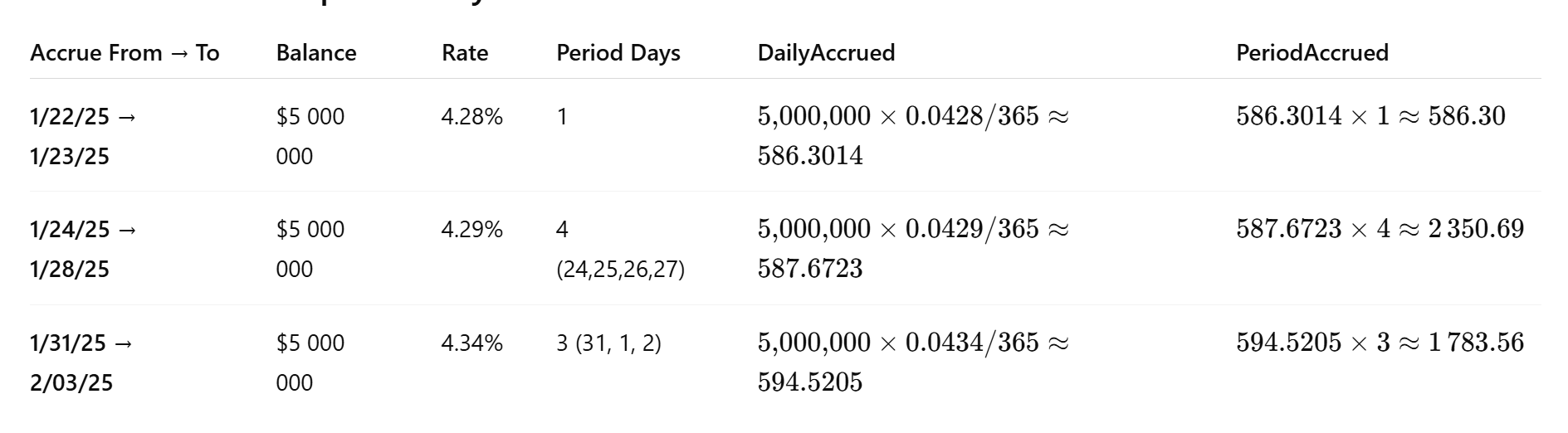
**Fetching rate based on this Business date(01/15/25):**

Now, in index, rate = 4.280000

**If this date would not be available in index, then the next lower date will be picked and the rate corresponding to that lower date.**



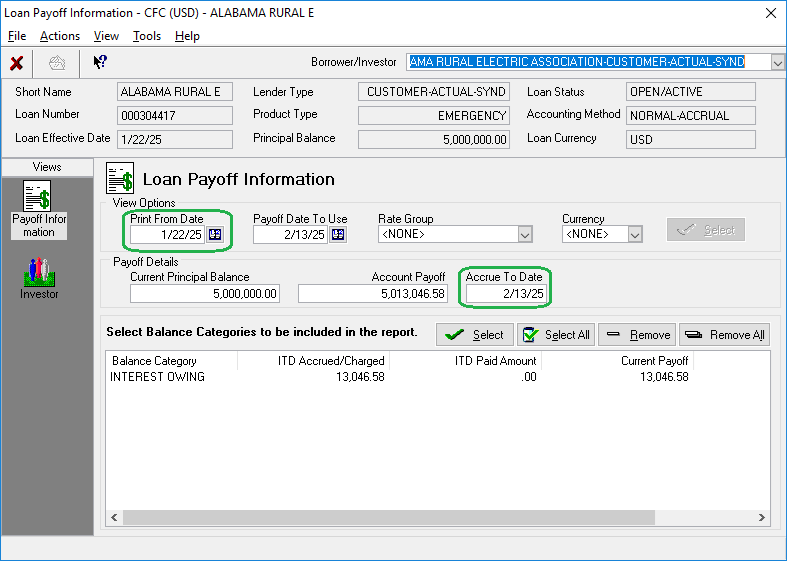




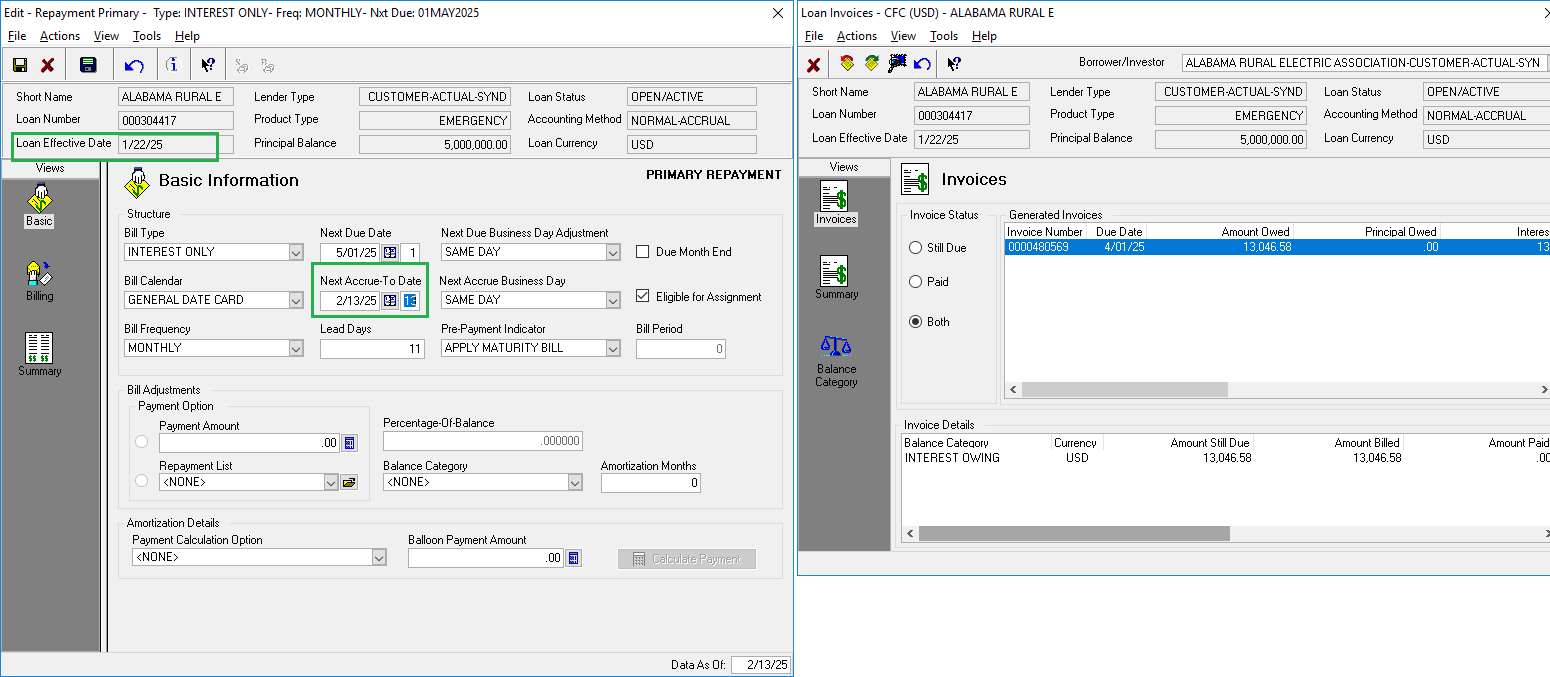
(RFR Calculation method is Simple)

**If you add up all the values in Period Accrued column, Total = 13 046.58**

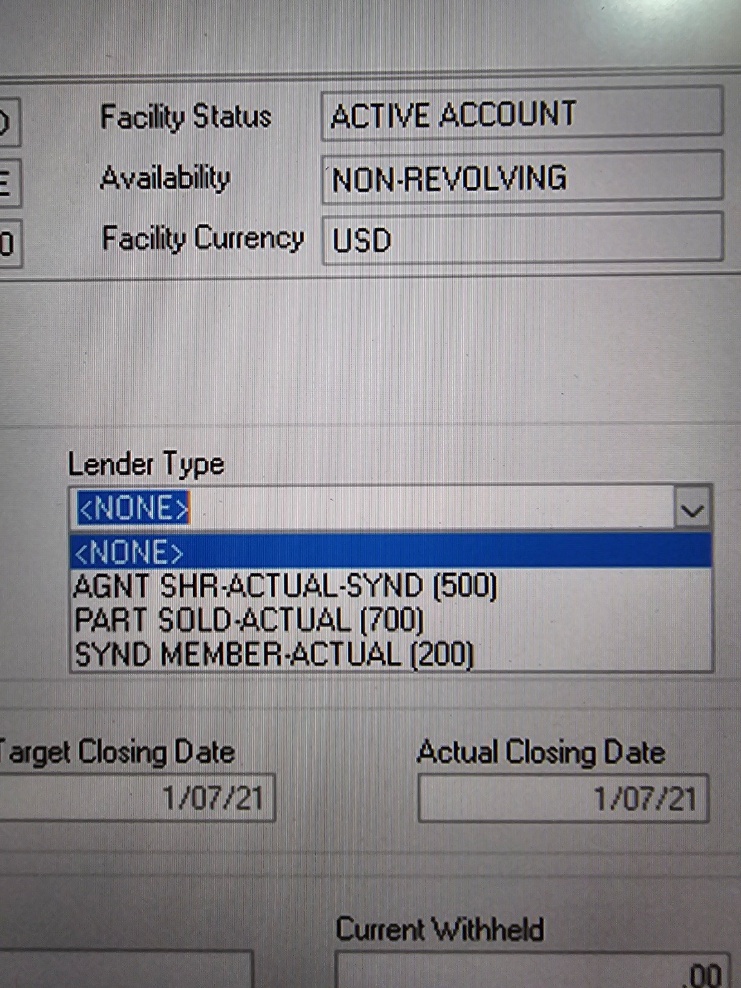
**That you can verify in Loan Payoff window:**

****

**The same you may verify in Loan Invoices:**



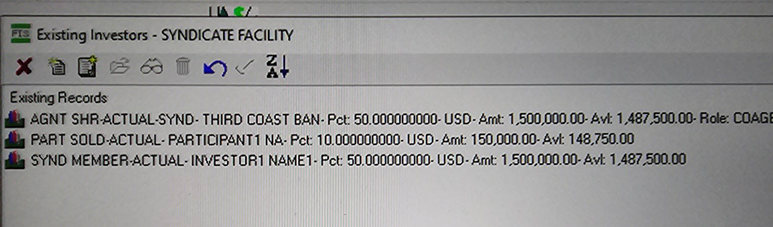
**Syndication Management**

500 = Agent, 200 = Syndicated Member, 700 = Participant

**Explaining "PART SOLD"**

“PART SOLD” refers to the portion of a loan that the original lender (**often the agent**) sells to a participant, meaning another financial institution. When we say "PART SOLD-ACTUAL," it denotes the actual amount sold. This term is tracked by code 700.

“Part Sold – Actual” (code 700) simply refers to the slice of the loan that your bank has sold down to a third-party participant. In other words, after the syndicate closes the deal, you can “sell” portions of your commitment to other investors (participants).

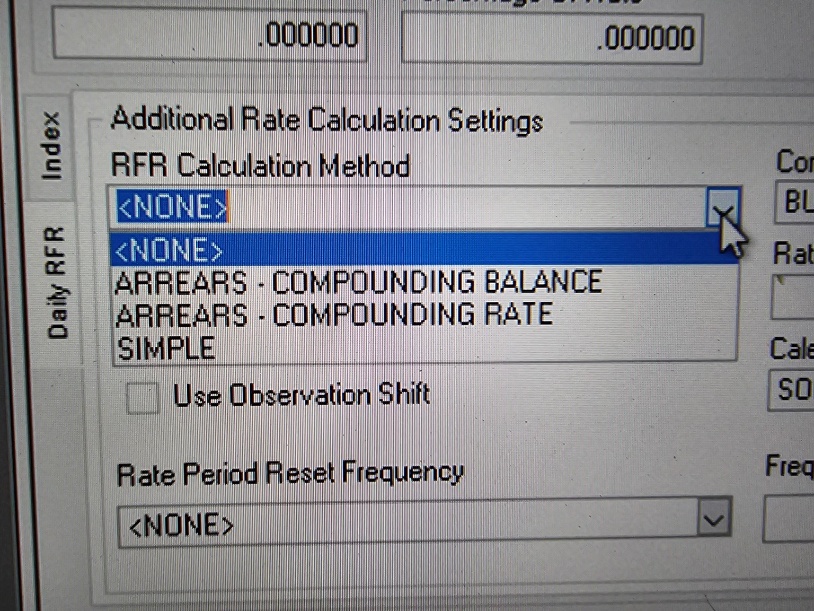


| **Line** | **Role** | **Pct of Facility** | **Amount (USD)** | **Available(USD)** | **What’s going on** |
| --- | --- | --- | --- | --- | --- |
| **AGNT SHR-ACTUAL-SYND – Third Coast** | Administrative Agent (code 500) | 50% | 1,500,000.00 | 1,487,500.00 | Third Coast Bank is your administrative agent and originally took on half of the loan. |
| **SYND MEMBER-ACTUAL – Investor1 Name1** | Syndication Member (code 200) | 50% | 1,500,000.00 | 1,487,500.00 | Investor1 is a core syndicate lender holding the other half of the commitment. |
| **PART SOLD-ACTUAL – Participant1 NA** | Loan Participant (code 700) | 10% | 150,000.00 | 148,750.00 | A €150 K slice of Third Coast’s original USD 1.5 M share has been sold down to Participant1. |

**Key points:**

1. **Total facility size = USD 3 million (AGNT: 1500000 + SYND MEM: 1500000)**
   * 50%/USD 1.5 M sits with the agent (Third Coast),
   * 50%/USD 1.5 M with the syndicate member (Investor1).
2. **PART SOLD (10%)** is carved out of the agent’s USD 1.5 M block. ACBS tracks it separately so that Participant1 receives its pro-rata interest & principal flows.
3. **“Available”** is simply the undrawn portion of each piece (after any disbursements).

By breaking the facility into Agent-Share, Member-Share and Part-Sold buckets, ACBS knows exactly who should get paid on each slice of the loan.

 **Simple** = no compounding.

 **Compounding-Rate** = compounds the **rate factors**, but still accrues on original principal in **one shot.**

 **Compounding-Balance** = actually steps the principal up each compounding period, so you literally earn “interest on interest” in your outstanding balance.

