Reflection

1) Which of the problems was the most challenging for you? What specific aspect puzzled you and how were you able to solve it (help from colleagues, notes, google search, office hours, generative AI, etc).

   Solving advanced problems 2 and 3 was very challenging and tough because their logic goes beyond basic finance and my background which I was not familiar with the formulas used. The understanding of the formula used to calculate the FV of annuity with an initial sum of money with monthly payouts was important. I used the corporate finance book, "berk jonathan and demarzo peter corporate finance", to clarify the mathematics, I used excel to validate each step and also referred to chatgpt for in-depth understanding and to fill conceptual gaps for this topic. The excel command used to calculate FV of an annuity is "=FV(rate, nper, pmt, pv)". As given below in the table is the FV of our initial sum after payouts.

PMT	600
PV	-10000
Rate	0.3%
NPER	FV
1	$9,430.00
2	$8,858.29
3	$8,284.86
4	$7,709.72
5	$7,132.85
6	$6,554.25
7	$5,973.91
8	$5,391.83
9	$4,808.01
10	$4,222.43
11	$3,635.10
12	$3,046.00
13	$2,455.14
14	$1,862.51
15	$1,268.09
16	$671.90
17	$73.91

I had to read book and documents to understand the formula behind this.

$$FV_k = PV\,(1+i)^k \;-\; PMT\,\frac{(1+i)^k - 1}{i}$$

I used the same formula to write my logic, what this formula does is that it first calculates the FV_of_our_PV for every month and subtracts the same from the FV_of_our_PMT to get the actual leftover FV post annuity payout.

I have also mentioned comments in my code for easier understanding of the logic. The second table given below describes how the FV of every month is being calculated with a snapshot from excel. Exactly same steps are being followed by the code to reach the final value.

t	PV_t = PV(1+r)**t	PMT_t = PMT*(((1+r)**t)-1)/r	FV = PV_t-PMT_t
Months	PV Compounded for each year	PMT Compunded each year	Final Future Value
1	($10,030.00)	$600.00	$9,430.00
2	($10,060.09)	$1,201.80	$8,858.29
3	($10,090.27)	$1,805.41	$8,284.86

4	($10,120.54)	$2,410.82	$7,709.72
5	($10,150.90)	$3,018.05	$7,132.85
6	($10,181.36)	$3,627.11	$6,554.25
7	($10,211.90)	$4,237.99	$5,973.91
8	($10,242.54)	$4,850.70	$5,391.83
9	($10,273.26)	$5,465.26	$4,808.01
10	($10,304.08)	$6,081.65	$4,222.43
11	($10,334.99)	$6,699.90	$3,635.10
12	($10,366.00)	$7,320.00	$3,046.00
13	($10,397.10)	$7,941.96	$2,455.14
14	($10,428.29)	$8,565.78	$1,862.51
15	($10,459.57)	$9,191.48	$1,268.09
16	($10,490.95)	$9,819.05	$671.90
17	($10,522.43)	$10,448.51	$73.91

The spreadsheet above shows the declining balance until the account can no longer fund a full $600 withdrawal. After **17 months**, $73.91 remains this matching my Python output and the output given for reference.

I realized that you can even solve this problem using calculating the monthly interest (interest = balance * monthly_rate) and the subtracting the payout from the balance and interest and saving everything back to balance (balance = balance + interest - monthly_payout). This logic also gives you the same result.

Advanced Question 3 was even tougher. I first had to understand the difference between simple depreciation and double depreciation. I used chatgpt to get a clear picture of the end cases and the year 0 printing. I have explained my logic in the code with comments and even attaching this handwritten algorithm to for better understanding. After solving these problems, I not only have gotten familiar with python functions and loops but also understand some essential finance concepts.

2) In which programs did you use loops? Were these count controlled loops or condition-controlled loops?

Program	Loop(s) present	Loop-type	Why
Basic Ques3	for i in range(sim):	**Count-controlled**	Because we have exact number of simulations.
Basic Ques4	while True: (breaks on a condition)	**Condition-controlled**	Runs until balance is less than lower bound.
Basic Ques5/6/7	for i in range(number of sales)	**Count-controlled**	Because we have to enter a user defined number of sales
Advanced Ques1	while True: (re-prompts until yes)	**Condition-controlled**	Loops until user enter something other than yes.
Advanced Ques2	while (FV > PMT):	**Condition-controlled**	Continues until the FV can't cover the next PMT.
Advanced Ques3	for year in range(life + 1):	**Count-controlled**	Iterates a predetermined number of years (life + 1). "+1" to account for year=0, where dep=0.

3) The advanced problems did not require the use of mainline functions with value returning functions. Describe how you would use such a structure in one of the advanced problems (no need to write code, just identify what the function would calculate and what it would return).

I have already written a returning value function for Advanced ques 1, I guess I can talk about others. If I keep the same code, since I have multiple values that had to be returned in order to be printed in the mainline function. Using the return

command in the helper function we can return as many variables as we want to use, but make sure that we assign those returned values to another variable in the in mainline.

Lets take the example of Ques 2, We can change the same code to a mainline driver that gathers user input and prints results from a value-returning helper (depreciation) that does all the calculations. We can return the number of months and balance using this code “return months, balance”. Like said, We have to make sure to save these in some other variable in the mainline code too. Use this code as an example, “months, residual = annuity_depletion(10_000, 600, 0.003)”. Similarly, we can do the same for question 3.