Intro2cs Ex3 – Retirement

Submission deadline

Thursday,07 November, 20:55

Objectives

In this exercise you will practice the use of functions, loops and working with files. You will also become familiar with data structures such as lists and tuples.

Ex4
Ex4 will be published during the next week, and its submission date is Nov 14th. Some of the issues in that exercise will be learned in the "tirgulim" of next week.

Planning for the future
One of the most debatable issues lately in Israel is the question of retirement accounts (Pension plans). As you embark on a career, retirement may seem very far away, but it is critical for each and every one of us. A big debate in the press is related to the question of how the retirement funds of our generation will look like. Assuming a larger life expectancy, our retirement money should provide for us for more years. We encourage you to read more and then compare information from different sources.
Although retirement may seem a long way off for you, we are going to explore some simple ideas in pension plans. Along the way, we are going to explore the use of successive approximation methods, as well as the use of some simple data structures, such as lists.

We will start with a simple model of saving for retirement. Let's assume that an employee earns a constant salary. Each year the employee saves some percentage of his salary into his retirement fund. Assume you are the worker and your starting yearly salary is represented by *salary*; the percentage of your salary you put into a retirement fund is *save*; and the annual growth percentage of the retirement fund is *growth_rate.* Then your retirement fund, represented by the list F, should increase as following:

	Retirement fund
End of year 1	F[0] = salary * save*0.01
End of year 2	F[1] = F[0]*(1+growth_rate*0.01)+salary*save*0.01
End of year 3	F[2] = F[1]*(1+growth_rate*0.01)+salary*save*0.01
…	

Assignment 1

Implement a function named *constant_pension* that receives four arguments: a salary, saving percentage (in percentage), an annual growth percentage for the investment account, and a number of work years. The function should return a list where each element in the list represents the value of the retirement fund in the end of a year, with the most recent year's value at the end of the list.

Complete the implementation of *constant_pension (salary, save, growth_rate, years)*

Write the code in the appropriate place in ex3.py template (you can find the link below). Make sure you've implemented the function as demanded, testing for bad inputs. To test your function, compare it with the school solution.

Mission 1 was relatively easy and simple; clearly the market does not grow at constant rate. A better model would be to account for variations in growth percentage each year.

Assignment 2

Implement `variable_pension`: This function takes *salary* and *save* as before, and *growth_rates* – a list of annual growth percentages in investments. The length of the list (*growth_rates*) will define the number of years you plan to work; *growth_rates* `[0]` is the growth rate for the first year, etc. (note that because the retirement fund's initial value is 0, *growth_rates* `[0]` is, in fact, irrelevant.). This function returns a list, whose values are the sizes of your retirement account at the end of each year (as in *constant_pension*).

Complete the implementation of *constant_pension (salary, save, growth_rates)*

Write the code in the appropriate place in ex3.py template.

But how should we choose where to invest the money? Well, most of us just go with the default suggested by the workplace, and end up with their retirement money spread between 10 different funds. Those of us who are smart, will try to understand what is suggested by each fund. Let's assume a clear future, where each fund promises you what will be the growing rate in each year.

Assignment 3

Implement *choose_best_fund(salary,save,funds_file)* In this function you will have to choose what is the best fund for your pension and return a tuple where its 1st value is the name of the fund and the 2nd is the value of the retirement account after retirement.

How to choose a best fund? The third parameter will be a path to a file with the exact details. The format of the file will be as following:

```
#nameOfFund0,annualGrowthYear0,annualGrowthYear1,annualGrowthYear2 …
#nameOfFund1,annualGrowthYear0,annualGrowthYear1,annualGrowthYear2 …
#nameOfFund2,annualGrowthYear0,annualGrowthYear1,annualGrowthYear2 …
…
```

The file contains the growth rate options of the different funds. You should go over all the funds and calculate the value of the account in its last year assuming the annual growth list of the specific fund. Choose the best fund - the best fund being the one that the value of it in the last year was the highest.
If two funds return the same value, return the first one.
You may assume that the form of the file is OK: Starts with #name and then a list of values separated by comma. If any error occur during the opening of the file (Can't find the filename, bad reading due to wrong types) - Let Python print its default error and out (Will happen automatically)

Complete the implementation of `choose_best_fund(salary,save,funds_file)`
Return a tuple containing the name of the chosen fund and the value of the retirement account in the last year *(fund_name,account_value)*
Write your code in the appropriate place in the ex3.py template.

An example for funds_file is published on the website (link below), you can try the school solution with it. But, note that we will test you with different *fund_files* - Your code must deal with different number of funds. The length of all funds (number of years) would be the same, but you won't know in advance this value. See more explanation in the template.

Assignment 4
In this assignment you will play the investment advisor. You are asked to implement two missions:

4.1 A potential investor asks you what will be the growth rate in a given year –

Implement `growth_in_year(growth_rates,year):`
Where year = 0 is the first value in the list. This function returns the value of `growth_rates` in the given year. In case of a year not in the list return None

4.2 Inflation matters. The market changed dramatically after inflation. Given a list of growth rates and inflation factors, one need to know what is the value of growth_list during the inflation. inflation should be adjusted for all years there is both an inflation factor and growth factor. inflation is defined as 100*((100+g)/(100+i)-1) where g is growth in that year and i is the inflation.
Implement `inflation_growth_rates(growth_rates,inflation_factors)` that returns a **NEW** list with the same length as `growth_rates` but during the inflation years the rates are adjusted.

For example if the original `growth_rates` were [2.0,2.0,2.0,2.0] and we call `inflation_growth_rates` (`growth_rates`, [1.0,1.0]) then the returned list will be [0.990099009900991, 0.990099009900991, 2.0, 2.0]. See school solution for more examples.
It is very important that you won't change the values in the original list and create a new one; we will test you for that.

After retirement, one must withdraw amount of money each year for living expenses.
Now, we want to model, how much money one can withdraw from his account after retirement. Once again we will assume simplified model – where there is no inflation at all so your annual expenses are constant – *expenses*. The complication here is that the money in your account will continue to growth according to `growth_rates`. To calculate what remain in your retirement fund each year after retirement we can use the following table:

	Retirement fund
End of year 1	F[0] = savings * (1+`growth_rates`[0]*0.01)-expenses
End of year 2	F[1] = F[0]*(1+`growth_rates`[1]*0.01)-expenses
End of year 3	F[2] = F[1]*(1+`growth_rates`[2]*0.01)-expenses
…	

Assignment 5

Implement *`post_retirement`*, which takes three arguments: an initial amount on your account, *`savings`*; a list of annual growth percentages on investments while you are retired (*`growth_rates`*), and your annual expenses (*`expenses`*). Assume that the increase in the investment account savings is calculated before subtracting the annual expenditures (as shown in the above table). Your function should return a list of the fund values on each year each year after retirement, accounting for annual expenses and the growth of the retirement fund. As in problem 2, the length of the *`growth_rates`* argument defines the number of years you plan to live after retirement.

Note that if the retirement fund balance becomes negative, expenditures should continue to be subtracted, and the growth rate comes to represent the interest rate on the debt (i.e. the formulas in the above table still apply).

Complete the definition for:
`post_retirement(savings, growth_rates, expenses)`:
Write your code in the appropriate place in the ex3.py template. compare it with the school solution

---

Dealing with errors and bad inputs

In this exercise you cannot assume that the input you've received from the user is correct. **You may assume that the type of each argument is correct but not its value**. In the ex3.py you will find specific details how to deal with errors, in each function, but generally speaking in case of bad input (wrong values) you should print nothing and return **None**.

Read thoroughly the coding style booklet from the course site. From this exercise and on you are required to follow the guidelines described in it.

Template file and an example of funds file

Template -ex3.py: http://moodle.cs.huji.ac.il/cs13/file.php/67101/ex3/ex3.py
Funds file example: http://moodle.cs.huji.ac.il/cs13/file.php/67101/ex3/file_0

School Solution

can be found in ~intro2cs/bin/ex3/ex3. It uploads a menu that is not demanded by your code. Your code only need to implement the functions. To enter a list write its length followed by values(L = [2,1,4] -> 3 2 1 4)
To input a file (choose_best_fund) use its full path
To test it on your own code, you can use the import function as we have learned in class.

Submission and Further Guidelines:

Submission:

1. Implement the 5 assignments in the ex3.py file
2. Create a tar file containing ex3.py and the README
3. Submit the file ex3.tar via the "Upload File" link on the course home page, under the ex3 link.
4. Make sure you pass the automatic tests

Running the tests yourself

- There are two ways you can run these testers and see the results without submitting your exercise and getting the pdf:
    a. The first – place your tar file in an empty directory, go to that directory using the shell and type: `~intro2cs/bin/testers/ex3 ex3.tar`

    a. The second – download a file - ex3testing.tar.bz2. Extract the contents (using tar –xjv ex3testing.tar.bz2) of the file and follow the instructions on the file named TESTING.