4CS001 - Coding Challenge 1

Functions, Variables and Expressions

Due: Sunday 25th October 2020 at 14:00

This assignment is worth 20% of the overall module grade

Introduction

This task will assess your knowledge of Python variables, expressions and string formatting. Make sure you've completed the proceeding workshop exercises before attempting this.

The marking scheme for this task is on Canvas, make sure you check back on a regular basis as you work through the assessment. **There is an additional challenge segment that can earn you extra credit, which you will need to complete to achieve the highest possible mark. Students who do not attempt this cannot achieve grades above 80% for this task.**

Getting Started

Start by downloading the file `compound_interest.py` from Canvas. Add your name and student number to the top of the file. Read the included documentation.

Task Overview

A new Wolverhampton based challenger bank called Wolving is launching in a few short months. They are in the process of building their website and have asked you to develop a program for prospective customers to show them how their savings could grow over time. Interest is usually paid on a yearly basis; however, the bank has asked that you incorporate compound interest to highlight their unique offerings when compared to competitors.

The idea behind compound interest is that the interest you earn each year is added to your principal (starting amount), so that the balance doesn't just grow, it grows at an increasing rate. This is one of the most useful concepts in finance. It is the basis of everything from personal saving plans to long term growth in the stock market and accounts for the effects of inflation. It is thought to have originated in the 17th century and can be thought of as 'interest on interest'.

The rate at which compound interest increases is determined by the number of compounding periods (the number of times interest is paid per year). Your software will showcase the effects of compound interest to potential customers, highlighting the fact that **Wolving pay interest quarterly**, compared to many other banks that simply pay interest on a yearly basis.

Calculating Simple and Compound Interest

Calculating yearly interest (simple interest) is straightforward, using the following formula:

$$\boldsymbol{A = P(1 + rt)}$$

On the other hand, calculating compound interest is a little more complicated:

$$\boldsymbol{A = P\left(1 + \frac{r}{n}\right)^{nt}}$$

There are clear similarities between them, both incorporate the same variables:

- $\boldsymbol{P}$ is the principal amount (the amount you start with).
- $\boldsymbol{r}$ is the annual rate of interest (as a decimal).
- $\boldsymbol{t}$ is the number of years the amount is invested.
- $\boldsymbol{A}$ is the amount at the end of the investment.

However, the compound interest formula also incorporates an additional variable $\boldsymbol{n}$. This is used to represent the number of compounding periods (interest payments per year).

Example Calculations:

Simple Interest:

£1500 invested over 6 years at 4.3% interest

1500 * (1 + 0.043 * 6) = 1887.00

Compound Interest:

£1500 invested over 6 years, compounding 4 times per year (quarterly) at 4.3% interest.

1500 * (1 + 0.043 / 4) $^{\mathbf{(4 * 6)}}$ **= 1938.84**

Program Requirements:

You will develop a program to calculate simple and compound interest. In order to do this, you will be required to implement several functions, specified in the template provided.

Once complete, your programs `main()` method should do the following:

1. Print a welcome message explaining the purpose of the program.
2. Prompt the user for the necessary inputs (see formulae and brief) and convert the values the user enters into suitable data types.
3. Perform the simple and compound interest calculations.
4. Print the results to the terminal in the format specified.

The next section will go over how to implement the individual functions in detail.

Constraints:

- Ensure that the interest rate is entered as a percentage and not a decimal.
- Ensure that all monetary values are formatted to two decimal places.

Extra Hints:

- Think about what data types are the most appropriate for each input value.
- Make sure you use parenthesis in your calculations.
- Review lecture two for more information on string formatting.
- Your programs output should be as close as possible to the example below.

Example Program Implementation (user input in red)

```
Welcome to the Wolving compound interest calculator.
This program calculates your potential returns when you invest with us

How much would you like to invest? 1500
What is the interest rate on your account? 4.3
How long are you planning to invest (in years)? 6
£1500 invested at 4.3% for 6 years is: £1887.00
£1500 invested at 4.3% for 6 years compounded 4 times per year is: £1938.84
```

Structure and Documentation

The structure of your code and documentation will be analysed and assessed.

This will be done using a static analysis tool called **Pylint**. This software checks the code in your program, ensures that it follows Python conventions and that all functions, classes and modules have been documented. You can read more about it here: https://www.pylint.org/.

Python has an official style guide named PEP8, which is where most Python conventions/coding standards originate from. Example checks that Pylint carries out to ensure that the PEP8 coding standard is followed include things such as:

- checking line-code's length
- checking if variable names are well-formed (snake case)
- checking if imported modules/functions are used
- checking if variables/function parameters are used

It is a good idea to run these checks on your code at regular intervals and before submitting. There are several free websites that allow you to do this e.g. https://pythonbuddy.com/.

Note: Marks will be deducted for warning and errors detected in your code.

Implementation Details

Step 1 – Implementing the *print_intro* Function:

To implement this function, you simply need to print an introduction to the user.

You should use the following message:

```
Welcome to the Wolving compound interest calculator.
This program calculates your potential returns when you invest with us
```

Step 2 – Implementing the *get_input* Function:

To implement this function, you need to request input from the user based on the variables used in the simple and compound interest calculations detailed above.

You will also need to covert the values the user enters into the correct type. Consider the most suitable data types for each, for instance, does it make sense to store years as a decimal?

Your function should return the values that the user entered, converted into the correct type.

Example Implementation:

```
How much would you like to invest? 1500
What is the interest rate on your account? 4.3
How long are you planning to invest (in years)? 6
```

Step 3 – Implementing the *simple_interest* Function:

To implement this function, you need to calculate the final value of an investment using the simple interest formula detailed above. It should take three parameters: the principal, interest rate and number of years the amount will be invested.

Your function should return the final value of the investment.

Example Implementation:

```
>>> simple_interest (1500, 4.3, 6)
1887.0
```

Step 4 – Implementing the *compound_interest* Function:

To implement this function, you need to calculate the final value of an investment using the compound interest formula detailed above. It should take three parameters: the principal, interest rate and number of years the amount will be invested.

Your function should return the final value of the investment.

Example Implementation:

```
>>> compound_interest (1500, 4.3, 6)
1938.8368221341054
```

Step 5 – Implementing the *print_output_compounding* Function:

To implement this function, you simply need to print formatted output to display the results of simple interest calculations so that it is clear and easy to understand. It should take four parameters, the principal, interest rate, number of years and the final investment value.

Example Implementation:

```
>>> print_output(1500, 4.3, 6, 1887.0)
£1500 invested at 4.3% for 6 years is: £1938.84
```

Step 6 – Implementing the *print_compounding_output* Function:

To implement this function, you simply need to print formatted output to display the results of compound interest calculations so that it is clear and easy to understand. It should take four parameters, the principal, interest rate, number of years and the final investment value.

Example Implementation:

```
>>> print_output(1500, 4.3, 6, 1938.8368221341054)
£1500 invested at 4.3% for 6 years compounded 4 times per year is: £1938.84
```

Challenge (Extra Credit)

Wolving believe that the compound interest calculator you've developed will help to draw in lots of new customers. However, they also feel it would be useful to allow users to enter a savings target and see how long it would take to reach their goal. Therefore, you've been asked to update your program to add this additional functionality.

To do this, you will need to implement the `calculate_years_to_target()`, `get_target_input()` and `print_target_output()` functions.

The first function `calculate_years_to_target()` should take three parameters, the principal, interest rate and savings target. It should calculate the number of years that it takes to reach the savings goal when compound interest is considered, before returning it.

The easiest way to go about this is to calculate the interest that should be paid for each individual compounding period, whilst keeping a running total. This will allow you to work out the number of compounding periods (and years) required to reach the savings goal.

Example Implementation:

```
>>> calculate_years_to_target(1500, 4.3, 2000)
7.0
```

Note: You should only count full years. For example, if it will take 5.1 years to achieve the savings goal, then your function should return the value 6.

The second function `print_target_output()` should take three parameters, the principal, interest rate and the number of years it takes to reach the savings goal.

Example Implementation:

```
>>> print_target_output(1500, 4.3, 7)
£1500.00 invested at 4.3% will allow you to reach your savings target in 7 years
```

The third function `get_target_input()` should function the same as the `get_input()` function, however, it should prompt users for an additional input – the savings goal amount.

Example Implementation:

```
>>> get_target_input()
How much would you like to invest? 1500
What is the interest rate on your account? 4.3
What is your savings goal? 2000
(1500.0, 4.3, 2000)
```

You will also need to update your main function to call the functions you've implemented. Users should be given the option to either calculate the simple and compound interest they could earn over time OR calculate the amount of time required to reach and savings goal.

Hints:

- You will need to be familiar with control structures to complete this challenge. These will be covered during the fourth lecture, feel free to do your own research.
- Interest should be distributed evenly across the course of a year. Therefore, if there are four payments windows in a year only ¼ of the interest is paid at each stage.

Example Implementation (user input in red)

```
Welcome to the Wolving compound interest calculator.
This program calculates your potential returns when you invest with us

Would you like to:
1. Calculate simple and compound interest over time
2. Calculate the amount of time required to hit a savings goal.
> 2
What is the principal amount? 1500
What is the rate? 4.3
What is your savings goal? 2000
£1500.00 invested at 4.3% will allow you to reach your goal in 6 years
```

```
Welcome to the Wolving compound interest calculator.
This program calculates your potential returns when you invest with us

Would you like to:
1. Calculate simple and compound interest over time
2. Calculate the amount of time required to hit a savings goal.
> 1
How much would you like to invest? 1500
What is the interest rate on your account? 4.3
How long are you planning to invest (in years)? 6
£1500 invested at 4.3% for 6 years is: £1887.00
£1500 invested at 4.3% for 6 years compounded 4 times per year is: £1938.84
```

Submission and Marking

You should upload your submission to Canvas. If you fail to do so, you will receive a grade of 0 NS (Non-Submission). The deadline for doing so is the 25th October 2020 at 14:00.

Students may submit work up to 7 calendar days after the published submission date in accordance with the University Late Submission & Extension Policy.

Students with a valid reason, as defined in the University Late Submission & Extension Policy and Procedure, may apply for an extension to the submission date of up to 7 calendar days.

Students without a valid reason, as defined in the University Late Submission & Extension Policy and Procedure, may submit work up to 7 calendar days after the published deadline but the mark will be subject to a penalty as follows;

a. Up to 2 days after the published deadline - a deduction of 10% of the maximum mark available from the actual mark achieved by the student.
b. After 2 days and up to 7 days after the published deadline - a deduction of 20% of the maximum mark available from the actual mark achieved by the student.

Your work will be automatically marked using a program that tests each of the individual functions you have implemented. Therefore, it is very important that you do not alter any of the function signatures in the template and implement everything as instructed.

You will receive an individualised test report, showing which of the tests passed and failed. Spot checks will be carried out to ensure that the marking is both fair and consistent. However, if you feel that you have been unfairly penalised, please do not hesitate to get in touch.