TASK

Capstone Project I — Variables and Control Structures

Visit our website

Introduction

WELCOME TO THE FIRST CAPSTONE PROJECT!

This Capstone Project is a milestone in your learning so far! In this task, you will be consolidating the knowledge you have gained and applying it to a real-world situation! This Capstone Project will allow you to demonstrate your competence in using variables, various data types and if-elif-else statements. It is worth spending time and effort to make this a project that you can be proud of! It could well be the first project that you add to your developer portfolio!

CAPSTONE PROJECTS

This is your first Capstone Project with Hyperion! Capstone Projects allow you to test your programming skills while creating a developer portfolio. It is essential to understand a programming language or technology for development. However, it is even more important to be able to apply your knowledge to create software that meets unique client specifications. Creating software allows you to highlight your development skills to a prospective employer!

As a software developer, you will need to demonstrate your ability to create software that is needed or wanted. You should also be able to improve existing software. Remember, any great design must be functional and meet the specifications provided by the user. A software solution that looks good and works, but doesn't do what the user wants it to, is like creating a bike with square wheels. A bike of this nature would only work in the scenario pictured below, showing a very specific design to demonstrate a physical principle in a science-teaching context, but it wouldn't be much good to get you from A to B. As a programmer, you need to aim to create resilient code that can function across a variety of contexts and that anticipates and plans for potential edge cases and unforeseen circumstances. Resilience testing is an important part of coding.

DEVELOPER PORTFOLIO

Developers who have the edge are those who find ways to apply their newfound skills from the get-go. A **developer portfolio** (a collection of software that you have made) allows you to demonstrate your skills rather than just telling people about them. It's a way of bringing your CV to life and introducing yourself to the world. As you learn more skills and put these into practice, each project that you complete will become more efficient and eye-catching.

This application series offers you the means to create the first project of your very own developer portfolio, allowing you to walk away from this course not only with a certificate but, more importantly, with a headstart into your career!

THE TASK AT HAND

In this Capstone Project, you will be creating a program that allows the user to calculate their interest on an investment or calculate the amount that should be repaid on a home loan each month.

Before you start developing this program, you will briefly learn a bit about interest. **Interest** occurs in almost all financial 'happenings', whether it be on a loan which ends up with you paying more to the bank, or on an investment which ends up with you earning more. There are two main types of interest: compound and simple interest.

Simple interest is continually calculated on the initial amount invested and is only calculated once per year. This interest amount is then added to the amount that you initially invested (known as the **principal amount**).

An example of this is if you invest R1000 at 10%, the first year you will earn R100 interest (R1000 × 0.10) giving you R1100. The next year the interest is still calculated on the principal amount (R1000) giving you another R100, making a total of R1200.

Compound interest is different in that the interest is calculated on the current total known as the **accumulated amount**.

To use the above example, imagine you invest R1000 at 10% compounded once a year. The first year you will earn R100, giving you an accumulated amount of R1100. In the second year, you will earn interest on the accumulated amount (1100 × 0.10), to earn R110 interest, giving you R1210.

If this doesn't make a huge amount of sense, don't worry too much. This is just a brief background to what you will be doing in the compulsory tasks. If you'd like to find out more, feel free to do a bit more research! However, the needed formulae are given in the task.

A note from our coding mentor
Valerie

Did you know that the programs that NASA used in the Apollo mission to the moon were less powerful than a pocket calculator? Yes, you read that right!

These ingenious computer systems were able to guide astronauts across 356 000 km of space from the Earth to the moon and return them safely. Today, a USB memory stick is more powerful than the computers that put man on the moon.

The Apollo Space Shuttle

Instructions

Feel free to refer back to previous material at any point or reach out for some assistance if you get stuck.

A key goal in this project is not only to get your code working according to the specifications provided but also to ensure that your code is readable. Readable code is easy to read and understand. Readable code is easier to maintain and troubleshoot. To make sure that your code is readable, do the following:

1. **Add comments:**

Recall a comment is information that you add to your code file that isn't read by the computer. A comment is not an instruction to the computer but rather information that clarifies code for human readers. Comments describe what the program does and why things are done as they are. Remember comments in Python start with the hash character (#).

Using appropriate comments can make code more readable. To illustrate, look at the code below:

```
year_born = int(input("Enter the year you were born: "))

if year_born >= 1946 and year_born <= 1964:
    print("You are classified as belonging to the Baby Boom Generation.")

elif year_born >= 1965 and year_born <= 1980:
    print("You are classified as belonging to Generation X.")

elif year_born >= 1981 and year_born <= 1996:
    print("You are classified as belonging to the Millennial Generation.")

elif year_born >= 1997 and year_born <= 2010:
    print("You are classified as belonging to Generation Z.")

else:
    print("You are age group does not have a classification.")
```

Look at the same code with added comments. See how comments can help? With comments, you don't have to spend much time trying to figure out what the code does. The comments tell you:

```
# This is an example Python program showing if-elif-else statements.
# The user inputs their birth year
# The program outputs the generation they belong to based on their birth
year
year_born = int(input("Enter the year you were born: "))

if year_born >= 1946 and year_born <= 1964:
    print("You are classified as belonging to the Baby Boom Generation.")

elif year_born >= 1965 and year_born <= 1980:
    print("You are classified as belonging to Generation X.")

elif year_born >= 1981 and year_born <= 1996:
    print("You are classified as belonging to the Millennial Generation.")

elif year_born >= 1997 and year_born <= 2010:
    print("You are classified as belonging to Generation Z.")

else:
    print("You are age group does not have a classification.")
```

As you may be able to see from the example below, too many comments can detract from the readability of your code instead of enhancing it. Be balanced when using comments. Add enough comments to assist another programmer using your code to see what is going on in your program quickly. The more lines of code you eventually have for a program, the more clear the need for comments will become.

```
# This is an example Python program showing if-elif-else statements.
# The user inputs their birth year
# The program outputs the generation they belong to based on their birth
year

# User enters birth year
year_born = int(input("Enter the year you were born: "))
# Baby boomer Generation 1946 to 1964
if year_born >= 1946 and year_born <= 1964:
    print("You are classified as belonging to the Baby Boom Generation.")
# Generation X 1965 to 1980
elif year_born >= 1965 and year_born <= 1980:
    print("You are classified as belonging to Generation X.")
```

```
# Millennial Generation 1981 to 1996
elif year_born >= 1981 and year_born <= 1996:
    print("You are classified as belonging to the Millennial Generation.")
# Generation Z 1997 to 2010
elif year_born >= 1997 and year_born <= 2010:
    print("You are classified as belonging to Generation Z.")
# No classification
else:
    print("You are age group does not have a classification.")
```

Read the **PEP8 style guide for comments** for commenting best practices. Be sure to add appropriate comments to all your code you write from now on.

2. **Use descriptive variable names:** Using meaningful names for variables also improves the readability of your code.

3. **Use whitespace and indentation** to enhance readability. Programs that have empty lines between units of work are easier to read.

4. Make sure that all output that your program provides to the user is easy to read and understand. Labelling all data that you output is essential to make the data your program produces more user-friendly. For example, compare the readability of the outputs in the images below. Notice how using spacing and labelling the output makes the second output much more user-friendly than the first:

   Output 1:

   ```
   Order number:266;Time ordered:18:03;Delivery type:Standard;City:Cape
   Town
   ```

   Output 2:

   ```
   ________________________________________
   Order number:   266
   Time ordered:   18:03
   Delivery type:  Standard
   City:           Cape Town
   ________________________________________
   ```

Compulsory Task 1

For this task, assume that you have been approached by a small financial company and asked to create a program that allows the user to access two different financial calculators: an investment calculator and a home loan repayment calculator.

- Create a new Python file in this folder called **finance_calculators.py**.
- At the top of the file include the line: `import math`
- Write the code that will do the following:

1. The user should be allowed to choose which calculation they want to do. The first output that the user sees when the program runs should look like this :

```
investment - to calculate the amount of interest you'll earn on your investment
bond       - to calculate the amount you'll have to pay on a home loan

Enter either 'investment' or 'bond' from the menu above to proceed:
```

2. How the user capitalises their selection should not affect how the program proceeds. i.e. 'Bond', 'bond', 'BOND' or 'investment', 'Investment', 'INVESTMENT', etc., should all be recognised as valid entries. If the user doesn't type in a valid input, show an appropriate error message.
3. If the user selects 'investment', do the following:

- Ask the user to input:
    - The amount of money that they are depositing.
    - The interest rate (as a percentage). **Only the number** of the interest rate should be entered — don't worry about having to deal with the added '%', e.g. The user should enter 8 and not 8%.
    - The number of years they plan on investing.
    - Then ask the user to input if they want "simple" or "compound" interest, and store this in a variable called *interest*. Depending on whether or not they typed "simple" or "compound", output the appropriate amount that they will get back after the given period, at the specified interest rate. See below for the formula to be used:

Interest formula:

The total amount when **simple interest** is applied is calculated as follows: $A = P(1 + r \times t)$
The Python equivalent is very similar: `A = P*(1 + r*t)`

The total amount when **compound interest** is applied is calculated as follows: $A = P(1 + r)^t$

The Python equivalent is slightly different: `A = P * math.pow((1+r),t)`

In the formulae above:

- 'r' is the interest entered above divided by 100, e.g. if 8% is entered, then r is 0.08.
- 'P' is the amount that the user deposits.
- 't' is the number of years that the money is being invested.
- 'A' is the total amount once the interest has been applied.

    - Print out the answer!
    - Try entering 20 years and 8 (%) and see what the difference is depending on the type of interest rate!

4. If the user selects 'bond', do the following:

- Ask the user to input:
    - The present value of the house. e.g. 100000
    - The interest rate. e.g. 7
    - The number of months they plan to take to repay the bond. e.g. 120

Bond repayment formula:

The amount that a person will have to be repaid on a home loan each month is calculated as follows: $repayment = \frac{i \times P}{1 - (1+i)^{-n}}$
The Python equivalent is slightly different:

```
repayment = (i * P)/(1 - (1 + i)**(-n))
```

In the formula above:

- 'P' is the present value of the house.

- ‘i’ is the monthly interest rate, calculated by dividing the annual interest rate by 12. Remember to divide the interest entered by the user by 100 e.g. (8 / 100) before dividing by 12.
- ‘n’ is the number of months over which the bond will be repaid.

    - Calculate how much money the user will have to repay each month and output the answer.

Thing(s) to look out for:

1. Before submitting your task, make sure that your code runs without any errors and that you have followed the guidelines in the instructions section of this document.

Rate us

Share your thoughts

Hyperion strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think the content of this task, or this course as a whole, can be improved, or think we’ve done a good job?

Click here to share your thoughts anonymously.