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 which 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!

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

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 important 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; not of any value!

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 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. 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 if you get stuck. Remember that if you require more assistance our mentors are always more than willing to help you!

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:** 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. Comments in Python start with the hash character, #.

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

```
rangeNum = int(input("Enter the max number you'd like to go up to: \n"))
for i in range (0, rangeNum):
        if i%2 == 0 :
                print(i)
```

   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.
# The user inputs a number.
# The program then outputs all the EVEN numbers from 0 to that number, using a for loop and if statement.

rangeNum = int(raw_input("Enter the max number you'd like to go up to: \n"))
for i in range (0, rangeNum):   #  Define a for loop that runs from 0 to the entered number.

        if i%2 == 0 : # This checks if the current number of the loop is EVEN.
                print i
```

As you may be able to see from the above example, 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.

Read this quick style guide for comments (here). 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. If needed, review the segment 'Introduction to variables' to review guidelines for creating good variable names.

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

4. **Look through this style guide** for best-practice guidelines regarding how your Python code should be written.

5. **Use a tool such as PyLint to check your code**. PyLint is an example of linting software. Linting software is used to check your code for errors and adherence to style guidelines. To use PyLint:
    a. Open the command line interface.

    In Windows, you can simply click the Start menu and type cmd in the search box to locate the Command Line. Alternatively, the Command Line should be one of the options under 'Programs' and you can simply click on the application to open it.

    

    With Mac OS, open the Command Line by opening the terminal. This can be done by opening the Applications folder, navigating to Utilities and then launching Terminal. Alternatively, you can search for "terminal" to find the application to launch.

    

b. Change directory to the folder that contains the code that you want to check by using the 'cd' command. E.g.

```
C:\>cd c:\example

c:\example>
```

c. Install PyLint typing the appropriate instruction, found here, into the command line.

d. Type 'pyLint' and the name of the python program that you want to check into the command line. E.g. `C:\example>pylint sample.py`

e. This should produce output that gives you feedback about where you can improve your code. E.g.

```
************* Module sample
sample.py:4:0: C0325: Unnecessary parens after 'while' keyword (superfluous-parens)
sample.py:5:0: C0325: Unnecessary parens after 'if' keyword (superfluous-parens)
sample.py:13:0: C0325: Unnecessary parens after 'if' keyword (superfluous-parens)
sample.py:14:0: C0325: Unnecessary parens after 'if' keyword (superfluous-parens)
sample.py:16:0: C0303: Trailing whitespace (trailing-whitespace)
sample.py:19:0: C0325: Unnecessary parens after 'if' keyword (superfluous-parens)
sample.py:33:0: C0325: Unnecessary parens after 'if' keyword (superfluous-parens)
sample.py:37:0: C0325: Unnecessary parens after 'if' keyword (superfluous-parens)
sample.py:41:0: C0325: Unnecessary parens after 'elif' keyword (superfluous-parens)
sample.py:43:0: C0325: Unnecessary parens after 'elif' keyword (superfluous-parens)
sample.py:53:0: C0303: Trailing whitespace (trailing-whitespace)
sample.py:55:0: C0303: Trailing whitespace (trailing-whitespace)
sample.py:1:0: C0114: Missing module docstring (missing-module-docstring)
sample.py:1:0: C0103: Constant name "allow" doesn't conform to UPPER_CASE naming style (invalid-name)
sample.py:2:0: C0103: Constant name "selection" doesn't conform to UPPER_CASE naming style (invalid-name)
sample.py:5:8: C0121: Comparison to False should be 'not expr' (singleton-comparison)
```

6. 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:

```
admin, Register Users with taskManager.py, Use taskManager.py to add the usernam
es and passwords for all team members that will be using this program., 10 Oct 2
019, 20 Oct 2019, No
```

Versus Output 2:

```
Task:  Register Users with taskManager.py
Assigned to: admin
Task description:       Use taskManager.py to add the usernames and passwords f
or all team members that will be using this program.
Date assigned:   10 Oct 2019
Due date:        20 Oct 2019
Task Complete?   No
```

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 :

```
Choose either 'investment' or 'bond' from the menu below to proceed:

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

   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.

2. 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 for.
- Then ask the user to input whether they want "simple" or "compound" interest, and store this in a variable called 'interest'. Depending on whether they typed "simple" or "compound", output the appropriate amount that they will get after the given period at the interest rate. Look below in "Interest formulae" for the formulae to be used.

Interest formulae:

Simple interest rate is calculated as follows: A = P(1 + r * t)
The Python equivalent is very similar: A =P*(1+r*t)

Compound interest rate 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' below 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 for.

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

3. 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 is each month is calculated as follows: repayment = PV\((1 - (1-(r/100))$^{-n}$)\r)

In the formulae above:

- 'PV' is the present value of the house.
- 'r' is the interest rate per period.
- '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. Make sure that you have installed and setup all programs correctly. You have setup **Dropbox** correctly if you are reading this, but **Python or IDLE** may not be installed correctly.
2. 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.