HyperionDev

Capstone Project – Variables and Control Structures

Task

Visit our website

Introduction

This capstone project is a milestone in your learning so far! In this project, you will be consolidating the knowledge you have gained and applying it to a real-world situation. It 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

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 learning 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.

Triciclo de rodas quadradas (Joseolgon, 2011)

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 course offers you the means to create projects for your very own developer portfolio, allowing you to walk away with not only 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 need to learn a bit about interest. As a developer in the workforce, you will need to research the concepts necessary to create programs, but as you are a beginner, we have done this for you and summarised below the concepts you need to understand to develop the program.

Interest occurs in almost all financial "happenings", whether it be on a loan that ends up with you paying more to the bank, or on an investment that 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**).
    - Example: If you invest 1000 Hyperion Dollars (H$) at 10% interest, the first year you will earn 100 H$ interest (1000 H$ × 0.10) giving you 1100 H$. The next year the interest is still calculated on the principal amount (1000 H$) giving you another 100 H$, making a total of 1200 H$.
- **Compound interest** is different in that the interest is calculated on the current total, known as the **accumulated amount**.
    - Example: Once again, imagine you invest 1000 H$ at 10% compounded once a year. The first year you will earn 100 H$, giving you an accumulated amount of 1100 H$. In the second year, you will earn interest on the accumulated amount (1100 H$ × 0.10), to earn 110 H$ interest, giving you 1210 H$.

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 project. If you'd like to find out more, feel free to do a bit more research! However, the required formulae are given in the project specifications.

Instructions

Feel free to refer back to previous material at any point if you get stuck. Remember that if you require further assistance, our reviewers 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, and is easier to maintain and troubleshoot. To make sure that your code is readable, do the following:

1. **Add comments**

   As you have already seen in the example files accompanying previous tasks, 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 Boomer 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("Your age group does not have a classification.")
```

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

```
# This is an example Python program showing if-elif-else statements.

# Get the user's birth year.
year_born = int(input("Enter the year you were born: "))

# Classify the user based on the user's birth year.
if year_born >= 1946 and year_born <= 1964:
    print("You are classified as belonging to the Baby Boomer 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.")

# Print a default message
else:
    print("Your 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 of them to assist another programmer using your code to quickly see what is happening in your program. The more lines of code you eventually have for a program, the more clear the need for comments will become.

```python
# 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("Your age group does not have a classification.")
```

Read the **PEP 8 style guide section on comments** for commenting best practices. Be sure to add appropriate comments to all the code you write from now on (this will be graded as part of the assessment of your submissions).

2. **Use descriptive variable names**

   Meaningful names for variables also improve the readability of your code.

3. **Use whitespace and indentation**

   Whitespace and indentation enhance readability. Programs that have empty lines between units of work are easier to read.

4. **Ensure output is readable**

   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
_____________________________________
```

Practical task Capstone Project

For this capstone project, 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.

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

    - How the user capitalises their selection should not affect how the program proceeds. In other words, “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.
    - If the user selects “investment”, 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 the following block for the formulae to be used.

Interest formulae:

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!

- If the user selects “bond”, ask the user to input:
  - The present value of the house, e.g., 100 000.
  - 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 repay 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.

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Take note

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

Share your thoughts

Please take some time to complete this short feedback **form** to help us ensure we provide you with the best possible learning experience.

Reference list

Joseolgon. (2011, November 24). *Triciclo de rodas quadradas*. Wikipedia. **https://pt.wikipedia.org/wiki/Roda_quadrada#/media/Ficheiro:Triciclo_de_Rodas_Quadradas.jpg**