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.