ASSIGNMENT 1

COMP202, Winter 2021

Due: Friday, Feb. 12^{th}, 11:59pm

Please read the entire PDF before starting. You must do this assignment individually.

Question 1:	50 points
Question 2:	50 points
	100 points total

It is very important that you follow the directions as closely as possible. The directions, while perhaps tedious, are designed to make it as easy as possible for the TAs to mark the assignments by letting them run your assignment, in some cases through automated tests. While these tests will never be used to determine your entire grade, they speed up the process significantly, which allows the TAs to provide better feedback and not waste time on administrative details. Plus, if the TA is in a good mood while he or she is grading, then that increases the chance of them giving out partial marks. :)

Up to 30% can be removed for bad indentation of your code as well as omitting comments, or poor coding structure.

To get full marks, you must:

- Follow all directions below
  - In particular, make sure that all file names, function names, and global variable names are **spelled exactly** as described in this document. Otherwise, a 50% penalty will be applied.
- Make sure that your code runs.
  - Code with errors will receive a very low mark.
- Write your name and student ID as a comment in all .py files you hand in
- Name your variables appropriately
  - The purpose of each variable should be obvious from the name
- Comment your work
  - A comment every line is not needed, but there should be enough comments to fully understand your program

Part 1 (0 points): Warm-up

Do **NOT** *submit this part, as it will not be graded. However, doing these exercises might help you to do the second part of the assignment, which will be graded. If you have difficulties with the questions of Part 1, then we suggest that you consult the TAs during their office hours; they can help you and work with you through the warm-up questions. You are responsible for knowing all of the material in these questions.*

Warm-up Question 1 (0 points)

Practice with Number Bases:

We usually use base 10 in our daily lives, because we have ten fingers. When operating in base 10, numbers have a `one`s column, a `ten`s column, a `hundred`s column, etc. These are all the powers of 10.

There is nothing special about 10 though. This can in fact be done with any number. In base 2, we have each column representing (from right to left) 1,2,4,8,16, etc. In base 3, it would be 1,3,9,27, etc.

Answer the following short questions about number representation and counting.

1. In base 10, what is the largest digit that you can put in each column? What about base 2? Base 3? Base n?
2. Represent the number thirteen in base 5.
3. Represent the number thirteen in base 2.
4. What is the number 11010010 in base 10?

Warm-up Question 2 (0 points)

Logic:

1. What does the following logical expression evaluate to?

```
(False or False) and (True and (not False))
```

2. Let a and b be boolean variables. Is it possible to set values for a and b to have the following expression evaluate as *False*?

```
b or (((not a) or (not a)) or (a or (not b)))
```

Warm-up Question 3 (0 points)

Expressions: Write a program `even_and_positive.py` that takes an integer as input from the user and displays on your screen whether it is true or false that such integer is even, positive, or both.

An example of what you could see in the shell when you run the program is:

```
>>> %Run even_and_positive.py
Please enter a number: -2
-2 is an even number: True
-2 is a positive number: False
-2 is a positive even number: False

>>> %Run even_and_positive.py
Please enter a number: 7
7 is an even number: False
7 is a positive number: True
7 is a positive even number: False
```

Warm-up Question 4 (0 points)

Conditional statements: Write a program `hello_bye.py` that takes an integer as input from the user. If the integer is equal to 1, then the program displays `Hello everyone!`, otherwise it displays `Bye bye!`.

An example of what you could see in the shell when you run the program is:

```
>>> %Run hello_bye.py
Choose a number: 0
Bye bye!

>>> %Run hello_bye.py
Choose a number: 1
Hello everyone!

>>> %Run hello_bye.py
Choose a number: 329
Bye bye!
```

Warm-up Question 5 (0 points)

Void Functions: Create a file called `greetings.py`, and in this file, define a function called `hello`. This function should take one input argument and display a string obtained by concatenating *Hello* with the input received. For instance, if you call `hello("world!")` in your program you should see the following displayed on your screen:

```
Hello world!
```

- Think about three different ways of writing this function.
- What is the return value of the function?

Warm-up Question 6 (0 points)

Void Functions: Create a file called `drawing_numbers.py`. In this file create a function called `display_two`. The function should not take any input argument and should display the following pattern:

```
  22
2   2
   2
 2
22222
```

Use strings composed out of the space character and the character '2'.

- Think about two different ways of writing this function.
- Try to write a similar function `display_twenty` which displays the pattern '20'

Warm-up Question 7 (0 points)

Fruitful Functions: Write a function that takes three integers `x`, `y`, and `z` as input. This function returns True if `z` is equal to 3 or if `z` is equal to the sum of `x` and `y`, and False otherwise.

Warm-up Question 8 (0 points)

Fruitful Functions: Write a function that takes two integers `x`, `y` and a string `op`. This function returns the sum of `x` and `y` if `op` is equal to `+`, the product of `x` and `y` if `op` is equal to `*`, and zero in all other cases.

Part 2

The questions in this part of the assignment will be graded.
The main learning objectives for this assignment are:

- Correctly create and use variables.
- Learn how to build expressions containing different type of operators.
- Get familiar with string concatenation.
- Correctly use `print` to display information.
- Understand the difference between inputs to a function and inputs to a program (received from the user through the function `input`).
- Correctly use and manipulate inputs received by the program from the function `input`.
- Correctly use simple conditional statements.
- Correctly define and use simple functions.
- Solidify your understanding of how executing instructions from the shell differs from running a program.

Note that the assignment is designed for you to be practicing what you have learned in class up to Tuesday, January 26. For this reason, **you are NOT allowed** to use what we have not seen in class (for example, you cannot use loops!). You will be heavily penalized if you do so.

For full marks on both questions, make sure to add the appropriate documentation string (docstring) to *all* the functions you write. The docstring must contain the following:

- The type contract of the function.
- A description of what the function is expected to do.
- At least 3 examples of calls to the function (beside the `display_welcome_menu` functions which always display the same content). You are allowed to use *at most* one example per function from this pdf.

Examples

At the end of the instructions for each question, we provide several **examples** of how your code should behave. All examples are given as if you were to call the functions from the shell.

When you upload your code to codePost, these examples will be run automatically to check that your code outputs the same as given in the example. However, **you must make sure your code/functions work for any inputs, not just the ones shown in the examples.** When the time comes to grade your assignment, we will run additional, private tests using inputs not seen in the examples.

Furthermore, please note that your code files for this question and all others **should not contain any function calls in the main body of the program** (i.e., outside of any functions). Code with function calls in the main body will automatically fail the tests on codePost and **be heavily penalized** if they remain in your final submission. It is OK to place function calls in the main body of your code for testing purposes, but if you do so, make certain that you remove them before submitting.

Question 1: Currency Exchange (50 points)

In this question, you will write a program that simulates a virtual currency exchange machine on the planet Orion IX. The citizens on the planet Orion IX use diverse forms of currency. Each city has its own currency, and in each city there can be many local currencies as well, e.g., for local sports teams, food dispensaries, and university courses. Machines to exchange between different currencies can be found everywhere. The particular machine in this question is located in a university of a popular city on the planet. The machine has one purpose: to convert between the local currency (dollars) to COMP202COIN, the currency of a popular course at the university, and vice versa. That is, you can put a certain amount of dollars or COMP202COIN into the machine, and the machine will output the corresponding amount in the opposite currency.

Note: Amounts of COMP202COIN are expressed in the hexadecimal (base 16) number system, which uses the digits 0 through 9 and also the letters A through F. That is, if someone says they have 4D2 COMP202COINs, they actually have 1,234 physical coins ($4D2_{16} = 1234_{10}$). When the user enters their amount of COMP202COIN into your program, they will be entering the amount in base 16.

Write your code for this question in a file called `currency_machine.py`. For full marks, all the following functions must be part of your program.

- `display_welcome_menu()`. This function takes no input and returns nothing. It displays a welcome message and a list with the options available to the customer:

```
>>> display_welcome_menu()
Welcome to the Orion IX COMP202COIN virtual exchange machine.
Available options:
1. Convert dollars into COMP202COIN
2. Convert COMP202COIN into dollars
3. Exit program
```

  You may personalize the welcome message, but the options available must be exactly the same as listed above.

- `get_solar_observation_fee(amount_of_comp202coin)`. This function takes one input, a string containing an integer in base 16 representing an amount of COMP202COINs. The planet Orion IX has two adjacent suns, and the currency exchange machine is solar powered. The current state of the suns is represented by two global variables `SUN1_SET` and `SUN2_SET` (booleans), which you should declare at the top of your file. You should set the first to False and the second to True. If either of the suns are still in the sky **(i.e., at least one of the two boolean global variables are `False`)**, then the function should return 0 as the machine can use the solar power and so there is no fee. Otherwise, if both suns have set **(i.e., both variables are `True`)**, then the machine must instead use a locally generated, costly form of power. In this case, the function should multiply the given number of COMP202COINs by the global variable `SOLAR_OBSERVATION_FEE_MULTIPLIER` (which you should declare at the top of your file to have the value 0.05), and return the result. **(Note that you cannot assume that the global variables will always be set to their default values as described above. That is, you must check their values in the function, you cannot just always return 0.)**

- `get_flat_fee()`. This function takes no inputs. It returns a global variable `COMP202COIN_FLAT_FEE` (which you should declare at the top of your file to have the value 10).

- `convert_COMP202COIN_to_dollars(amount_of_comp202coin)`. This function takes one input, a string containing an integer in base 16 representing an amount of COMP202COINs. It should convert the given amount of COMP202COINs into dollars, using the following formula, and then return the number of dollars as a float.

  Amount of dollars = (Amount of COMP202COINs in base 10 * COMP202COIN-Dollar exchange rate) - (**Solar observation fee multiplier** * Amount of COMP202COINs in base 10) - Flat fee

The COMP202COIN-Dollar exchange rate should be specified by a global variable at the top of your program called `COMP202COIN_DOLLAR_EXCHANGE_RATE`. Set it to be 0.05. That is, for every COMP202COIN, the user would receive 0.05 dollars. **(Note that it is possible for this function to return a negative amount of dollars, depending on the values used in the formula above.)**

Note: Some of the different parts of the formula above will come from the functions mentioned above. For example, the term (`Solar observation fee multiplier * Amount of COMP202COINs in base 10`) should be calculated solely by calling your `get_solar_observation_fee()` function.

- `convert_dollar_to_COMP202COIN(amount_of_dollars)`. This function takes one input, a string containing an float (base 10) representing an amount of dollars. It should convert the given amount of dollars into COMP202COINs, using the following formula, and then return the number of COMP202COINs as a string containing an integer in base 16. (If the formula results in a decimal number, the number should be rounded down to the nearest integer, as the machine can only output whole COMP202COINs and cannot cut them into pieces.)

  Amount of COMP202COINs in base 10 = Amount of dollars * Dollar-COMP202COIN exchange rate

  The Dollar-COMP202COIN exchange rate should be specified by a global variable at the top of your program called `DOLLAR_COMP202COIN_EXCHANGE_RATE`. Set it to be 0.01. That is, for every dollar, the user would receive 0.01 COMP202COINs.

  Note that there is no flat fee nor solar observation fee to convert from dollars to COMP202COIN, because the customer is already getting a bad deal.

  Note: To convert a base 10 number into a base 16 number, you can use the built-in `hex(number)` function, which returns `number` in base 16 in a string.

- `get_excess_dollars_after_conversion(amount_of_dollars)`. This function takes one input: a float (base 10) representing an amount of dollars. It returns the amount of dollars (as a float) that are left over after converting the amount to COMP202COIN and rounding down. For example, trying to convert 123 dollars to COMP202COIN (with the above exchange rate of 0.01) would result in 1 COMP202COIN, with 23 dollars left over. In this case, the function should return 23.

  Further, the machine can only give COMP202COINs to the user if it has enough in stock. Create a global variable `COMP202COIN_SUPPLY` at the top of your program and set it to the string '64' (a base 16 number). If the amount of dollars given to the function is larger than the supply, then the function should return the entire amount of dollars (as it is all considered excess).

- `operate_machine()`. This function takes no inputs and returns no values. The function performs the following tasks in the following order:
  - It displays the menu of the machine.
  - It retrieves an integer from the user indicating their choice.
  - If the user asked to end the program, the program terminates.
  - Otherwise, it retrieves the amount of the currency that the user wishes to convert.
  - It converts the amount into the other currency and displays five pieces of information: the amount the user entered and what currency it was, the amount the user will receive and what currency it is/they are, and, if dollars are being converted to COMP202COINs, the amount that was not able to be converted. E.g.: `'You have deposited [x] dollars. You will receive [y] COMP202COIN, and [z] dollars will be returned to you.'` If the entire **(total)** amount **entered by the user** was not able to be converted, then the program should instead print `'Your transaction cannot be completed due to insufficient funds.'` instead of the above message.

Note that COMP202COIN values should be printed in base 16, and dollar amounts should be rounded to two decimal places.

As long as you maintain the correct meaning, you are free to personalize the messages displayed by your vending machine.

For full credit you should never be repeating code, but rather calling helper functions! You are welcome to add additional functions if you think this can increase the readability of your code.

Examples (as executed in Thonny)

Below you can find a couple of examples of interactions with the vending machine. Note that the inputs provided from the user appear highlighted in grey.

Example 1:

```
Welcome to the Orion IX COMP202COIN virtual exchange machine.
Available options:
1. Convert dollars into COMP202COIN
2. Convert COMP202COIN into dollars
3. Exit program
> 1
Enter the amount of dollars to convert: 500
You have deposited 500.0 dollars. You will receive 0x5 COMP202COIN,
and 0.0 dollars will be returned to you.
```

Example 2:

```
>>> operate_machine()
Welcome to the Orion IX COMP202COIN virtual exchange machine.
Available options:
1. Convert dollars into COMP202COIN
2. Convert COMP202COIN into dollars
3. Exit program
> 1
Enter the amount of dollars to convert: 599
You have deposited 599.0 dollars. You will receive 0x5 COMP202COIN,
and 99.0 dollars will be returned to you.
```

Example 3:

```
>>> operate_machine()
Welcome to the Orion IX COMP202COIN virtual exchange machine.
Available options:
1. Convert dollars into COMP202COIN
2. Convert COMP202COIN into dollars
3. Exit program
> 1
Enter the amount of dollars to convert: 101010
Your transaction cannot be completed due to insufficient funds.
```

Example 4:

```
>>> operate_machine()
Welcome to the Orion IX COMP202COIN virtual exchange machine.
Available options:
1. Convert dollars into COMP202COIN
2. Convert COMP202COIN into dollars
3. Exit program
> 2
Enter the amount of COMP202COIN to convert: CFF
You have deposited CFF COMP202COIN. You will receive 156.35 dollars.
```

Question 2: Pizza Calculator (50 points)

Last week, Johnny went to a pizza shop and asked for a 12-inch pizza. The shop owner said they didn't have any 12-inch pizzas left and suggested selling Johnny two 6-inch pizzas for the same price. Johnny accepted the offer.

After Johnny got home, his girlfriend told him that he had been ripped off. The area of a circle is

$$Area = \pi r^2$$

Where r is the radius of the circle and π is roughly 3.14. So, the 12-inch pizza, with a radius of 6 inches, has an area of

$$3.14 * 6^2 = 113.04$$

whereas a 6-inch pizza, with a radius of 3 inches, has an area of

$$3.14 * 3^2 = 28.26$$

This means that two 6-inch pizzas only have an area of $28.26 * 2 = 56.52$! Thus, two 6-inch pizzas are equivalent to only half of a 12-inch pizza!

Feeling slightly traumatized by this experience, Johnny decides to hire you to write a Python program to prevent being ripped off again by unethical pizza shops.

For this question, create a program called `pizza_calculator.py` and write the following functions:

- `display_welcome_menu`: This function takes no input and *displays* a welcoming message as well as a list with all the options available to Johnny (the user).

  That is,

  ```
  >>> display_welcome_menu()
  Welcome to the COMP 202 fair pizza calculator!
  Please chose one of the following modes:
  A. "Quantity mode"
  B. "Price mode"
  ```

- `get_fair_quantity`: This function takes two positive integers as input representing the diameters of two pizzas. The function *returns* an integer indicating the minimum number of smaller pizzas Johnny must order in order for him to get *at least* the same amount of pizza as one large pizza.

  For example:

  ```
  >>> get_fair_quantity(14, 8)
  4
  >>> get_fair_quantity(8, 14)
  4
  >>> get_fair_quantity(14, 5)
  8
  >>> get_fair_quantity(5, 5)
  1
  ```

- `get_fair_price`: This function takes as input an integer indicating the diameter of the large pizza, a real number indicating the price of the large pizza, an integer indicating the diameter of the small pizza, and one last integer indicating the number of small pizzas to be ordered. You can assume the inputs will always be positive numbers in the order just specified. The function *returns* the total price Johnny should be paying to buy the smaller pizzas such that the amount of pizza per dollar is the same as that of the larger pizza. The return value should be a real number with no more than 2 digits after the decimal point.

For example:

```
>>> get_fair_price(12, 10.0, 6, 2)
5.0
>>> get_fair_price(14, 9.55, 8, 4)
12.47
```

- `run_pizza_calculator`: This function takes no inputs and returns no value. The function performs the following tasks in this following order:
  - It displays the menu of the pizza calculator
  - It retrieves a string indicating the choice of the user.
  - If "Quantity mode" was selected, then it retrieves two more integers indicating the diameter of the large and the small pizza. It then display the quantity of small pizzas the user should buy to be satisfied.
  - If "Price mode" was selected, then it retrieves the following:
    1. The size (diameter) of the larger pizza (integer)
    2. The price of the larger pizza (real number)
    3. The size (diameter) of the smaller pizza (integer)
    4. The number of smaller pizzas (integer)

    It then displays the fair price the user should pay for the specified number of small pizzas.
  - If any other mode is selected, the function should display a message indicating that such mode is not supported.

  You can find below a couple of example of executing the function. Note that the inputs provided from the user appear in a different font and in blue.

```
>>> run_pizza_calculator()
Welcome to the COMP 202 fair pizza calculator!
Please chose one of the following modes:
A. "Quantity mode"
B. "Price mode"

Enter your choice: 1

This mode is not supported


>>> run_pizza_calculator()
Welcome to the COMP 202 fair pizza calculator!
Please chose one of the following modes:
A. "Quantity mode"
B. "Price mode"

Enter your choice: a

This mode is not supported
```

```
>>> run_pizza_calculator()
Welcome to the COMP 202 fair pizza calculator!
Please chose one of the following modes:
A. "Quantity mode"
B. "Price mode"

Enter your choice: A

You selected "Quantity mode"

Enter the diameter of the large pizza: 14
Enter the diameter of the small pizza: 8

To be fully satisfied you should order 4 small pizzas


>>> run_pizza_calculator()
Welcome to the COMP 202 fair pizza calculator!
Please chose one of the following modes:
A. "Quantity mode"
B. "Price mode"

Enter your choice: B

You selected "Price mode"

Enter the diameter of the large pizza: 12
Enter the price of the large pizza: 10.0
Enter the diameter of the small pizza: 6
Enter the number of small pizzas you'd like to buy: 2

The fair price to pay for 2 small pizzas is $5.0


>>> run_pizza_calculator()
Welcome to the COMP 202 fair pizza calculator!
Please chose one of the following modes:
A. "Quantity mode"
B. "Price mode"

Enter your choice: B

You selected "Price mode"

Enter the diameter of the large pizza: 14
Enter the price of the large pizza: 9.55
Enter the diameter of the small pizza: 8
Enter the number of small pizzas you'd like to buy: 4

The fair price to pay for 4 small pizzas is $12.47
```

For full credit you should never be repeating code, but rather calling helper functions! You are welcome to add additional functions if you think this can increase the readability of your code.

What To Submit

You must submit all your files on codePost (https://codepost.io/). The file you should submit are listed below. Any deviation from these requirements may lead to lost marks.

`currency_machine.py`
`pizza_calculator.py`
`README.txt` In this file, you can tell the TA about any issues you ran into doing this assignment. If you point out an error that you know occurs in your program, it may lead the TA to give you more partial credit.
Remember that this assignment like all others is an `individual` assignment and must represent the entirety of your own work. You are permitted to verbally discuss it with your peers, as long as no written notes are taken. If you do discuss it with anyone, please make note of those people in this `README.txt` file. If you didn't talk to anybody nor have anything you want to tell the TA, just say "nothing to report" in the file.