ASSIGNMENT 1

COMP202, Fall 2020

Due: Wednesday, October 7th, 11:59pm

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

Question 1:	35 points
Question 2:	35 points
Question 3:	30 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 and function 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 at the top of 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 the videos up to and including Week 4.3 (Docstrings). For this reason, you are NOT allowed to use anything seen after Week 4.3 or not seen in class at all (for example, you cannot use loops, as they were shown in the video of Week 4.4 and beyond). You will be heavily penalized if you do so.

For full marks on the following three questions, in addition to the points listed on page 1, 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 (except when the function has only one possible output, in which case you can provide only one example). 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 that may use 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 that does not conform in this manner will automatically fail the tests on codePost and **be heavily penalized**. 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: Smoothie Ordering (35 points)

With the rise in remote learning, many classrooms in campuses across the country are currently vacant. To capitalize on this situation, a nearby school called MacGill University is converting some classrooms into temporary food delivery service facilities. One such company called Smooth Smoothies Ltd (SSL) has moved into the computer science building of said university. Smoothies are prepared in the building, and delivered to clients who order through an online system.

You have been contracted by Smooth Smoothies Ltd to develop an ordering system for their smoothies using Python. Users will enter their order (smoothie type, size and topping) through the Python shell, and receive the total amount payable with appropriate tax amounts listed.

Ordering will proceed as follows:

1. The customer will be welcomed to Smooth Smoothies smoothie system for ordering smoothies.
2. The customer will be asked which smoothie they would like. There will be four options from which to choose. Each option will include the name and base price of the smoothie.
3. The customer will be asked which size of smoothie they would like. There will be four options from which to choose. Each option will include the size and any additional cost (the default size will be medium and will have no additional cost).
4. The customer will be asked which topping they would like. There will be four options from which to choose. The user will only be able to choose one topping, and all toppings will be $1. The user may decide to select 'no topping' (which will be one of the four options).
5. The customer will be provided a receipt showing the subtotal, GST, QST, and total amount payable.

You will write your code for this question in a file called `smoothie.py`.

To begin, you will define each of the four smoothie types, sizes, and toppings in global variables at the top of the code file. Each smoothie type, size and topping has a name and a cost. For example, there may be a size named 'large' with cost '3'. The global variables must have the name `Yn_NAME` and `Yn_COST`, where `Y` is one of `SMOOTHIE`, `SIZE`, or `TOPPING`, and n is a number from 1 to 4 (for example, there will be global variables called `TOPPING1_NAME` and `TOPPING1_COST`). The name will be a string and the cost will be a floating-point number. You will then use these global variables later in your code to refer to the names and costs of the smoothie types, sizes and toppings.

SSL leaves it to your discretion to come up with names and costs for the smoothie types and toppings. (If you wish, you may use the names provided in the example on the next page.) However, one of the topping options must be 'no topping' with a cost of $0. Further, one of the four smoothies must be named as described on the next page.

The size names and prices must be as follows:

- Small: -$2 to base price
- Medium: $0 (no additional cost)
- Large: +$2
- Galactic: +$100

You will then implement the ordering system by writing the following functions. Remember to use the global variables discussed above when necessary.

- `pose_question_with_costs(question, option1, cost1, option2, cost2, option3, cost3, option4, cost4)`: This function takes 9 arguments: one string representing a question, then 4 strings and 4 floats representing the potential answers to the question and their respective costs. The function must present the `question` to the user and display the four options and costs, with each option and associated cost on its own line, and the number of the option (1 through 4) at the start of each option line. Tabs should separate the option number, cost, and option. The user should then be prompted for input, and the option name corresponding to their entered number should be returned (e.g., if they entered 2, then `option2` should be returned). You cannot assume anything about the input the user may provide; if they did not enter a number between 1 and 4, then return the empty string. Otherwise, before the function returns, make sure to print out `You have selected [option name]` to acknowledge that the user entered a valid option.

- `calculate_subtotal(smoothie_type, smoothie_size, topping)`: This function will calculate and return the price of the smoothie as a floating point number, given the three parameters (smoothie type, size and topping, all strings). The price should be rounded to two decimal places.
- `print_receipt(subtotal, smoothie_type, smoothie_size, topping)`: This function takes one float and three strings. It will print out the customer's receipt, as follows. First, it will print out their order as: `You ordered a [size] [type] smoothie with [topping].` (If they did not order a topping, do not print `with [topping]`.) Then, it will display the subtotal, GST (9.975%), QST (5%), and final total. (Tax will be calculated on top of the subtotal.) Each of the four amounts should be displayed on a different line, and amounts should be rounded to two decimal places. The function will then return the final total as a float.
- `order`: This function takes no inputs. It will welcome the user to the smoothie ordering system, then call the other functions as necessary in order to obtain the user's order and show them their receipt. If the user enters anything invalid (thus making the `pose_question` function return the empty string), abort the order by telling the user that they have entered an invalid input.

  The `order` function is also where you must implement a special directive issued by SSL research division. They have recently created a new smoothie flavour called 'Onion Toffee' (with a cost of $9.99) and want to encourage everyone to try it. Therefore, after greeting the user to the system, make sure to mention this new smoothie. Further, to the same end, SSL has decided not to buy ingredients for any other type of smoothie. If the user selects any smoothie type other than 'Onion Toffee', then the `order` function must inform them that that smoothie is out of stock, and that they will receive an 'Onion Toffee' smoothie instead.

Examples (as executed in Thonny)

Example 1:

```
>>> order()
Welcome to Smooth Smoothies Smoothie Ordering System
Have you tried our new Onion Toffee smoothie?
Which smoothie would you like?
1)       $ 4.99         Pineapple Banana
2)       $ 6.49         Almond Basil
3)       $ 0.99         Purple Surprise
4)       $ 9.99         Onion Toffee
Your choice (1-4): 1
You have selected Pineapple Banana.
Unfortunately, we are out of Pineapple Banana
You will be served Onion Toffee smoothie.
Which size would you like?
1)       $ -2         small
2)       $ 0          medium
3)       $ 2          large
4)       $ 100        galactic
Your choice (1-4): 1
You have selected small
Which topping would you like?
1)       $ 0         no topping
2)       $ 1         cinnamon
```

```
3)       $ 1          chocolate shavings
4)       $ 1          shredded coconut
Your choice (1-4): 1
You have selected no topping
You ordered a small Onion Toffee smoothie
Smoothie cost: $7.99
GST:    $0.80
QST:    $0.40
Total:  $9.19
```

Example 2:

```
>>> order()
Welcome to Smooth Smoothies Smoothie Ordering System
Have you tried our new Onion Toffee smoothie?
Which smoothie would you like?
1)       $ 4.99         Pineapple Banana
2)       $ 6.49         Almond Basil
3)       $ 0.99         Purple Surprise
4)       $ 9.99         Onion Toffee
Your choice (1-4): 3
You have selected Purple Surprise.
Unfortunately, we are out of Purple Surprise
You will be served Onion Toffee smoothie.
Which size would you like?
1)       $ -2          small
2)       $ 0           medium
3)       $ 2           large
4)       $ 100         galactic
Your choice (1-4): 4
You have selected galactic
Which topping would you like?
1)       $ 0          no topping
2)       $ 1          cinnamon
3)       $ 1          chocolate shavings
4)       $ 1          shredded coconut
Your choice (1-4): 4
You have selected shredded coconut
You ordered a galactic Onion Toffee smoothie with shredded coconut
Smoothie cost: $110.99
GST:    $11.07
QST:    $5.55
Total:  $127.61
```

Example 3:

```
>>> order()
Welcome to Smooth Smoothies Smoothie Ordering System
Have you tried our new Onion Toffee smoothie?
Which smoothie would you like?
1)        $ 4.99          Pineapple Banana
2)        $ 6.49          Almond Basil
3)        $ 0.99          Purple Surprise
4)        $ 9.99          Onion Toffee
Your choice (1-4): nine thousand and one
Sorry, that is not a valid option.
```

Example 4:

```
>>> x = pose_question_with_costs("Which size would you like?", "small", -2.0, "medium", 0.0,
                                 "large", 2.0, "galactic", 100.0)
Which size would you like?
1)        $ -2.0          small
2)        $ 0.0          medium
3)        $ 2.0          large
4)        $ 100.0          galactic
Your choice (1-4): 3

>>> print(x)
'large'
```

Example 5:
Note: This example assumes that the global variables below have already been defined at the top of the code file with associated global variables for the costs of $4.99, $100, and $1, respectively.

```
>>> calculate_subtotal(SMOOTHIE1_NAME, SIZE4_NAME, TOPPING4_NAME)
105.99
```

Example 6:
Note: This example assumes that the global variables below have already been defined at the top of the code file with associated global variables for the costs of $4.99, $100, and $1, respectively, and that the names were as given in the previous examples.

```
>>> total = print_receipt(105.99, SMOOTHIE1_NAME, SIZE4_NAME, TOPPING4_NAME)
You ordered a galactic Pineapple Banana smoothie with shredded coconut
Smoothie cost: 105.99
GST:         $10.57
QST:         $5.30
Total:         $121.86

>>> print(total)
121.86
```

Question 2: Shipping Books (35 points)

In this question you will write a variety of functions to aid in preparing books for shipment from a book depot. You will write your code for this question in a file called `shipment.py`.

Before books are placed into a box for shipment, their ISBNs are scanned for record purposes. ISBN stands for International Standard Book Number and consists of a series of digits that uniquely identifies a book. Today, books have 13-digit ISBN numbers, but we will consider the older 10-digit numbers for simplicity. The important thing to know about an ISBN number is that its right-most (10th) digit is known as a 'checksum.' This checksum can be calculated by performing a computation on the other 9 digits. Each time an ISBN is scanned, this calculation is performed, and the result of the calculation is checked against the actual 10th digit. (If they do not match, then it is assumed that the ISBN is damaged in some way.)

The checksum for a 10-digit ISBN number is calculated as follows:

$(x_1 + 2 * x_2 + 3 * x_3 + ... + 9 * x_9) \% 11$

where each x_i is digit i of the ISBN (from left to right – x_1 is the left-most digit). The result of this calculation is the checksum, unless equal to 10, in which the checksum is instead a capital 'X'.

We will write some functions to calculate the checksum of an ISBN number and determine if an ISBN is valid or not.

- `calculate_isbn_checksum_by_digits(d1, d2, d3, d4, d5, d6, d7, d8, d9)`: This function takes in a 10-digit ISBN as 9 one-digit integers, and calculates and returns the checksum as a string.
- `calculate_isbn_checksum(isbn)`: This function takes as input a 9-digit integer representing the 9 left-most digits of an ISBN, and calculates and returns the checksum as a string.
- `is_isbn(isbn, checksum)`: This function takes two arguments: a 9-digit integer representing the 9 left-most digits of an ISBN, and a checksum (string), and returns `True` if the ISBN is valid (i.e., if the given checksum matches the calculated checksum) and `False` otherwise.

Note that you cannot convert the ISBN integers into string type for these functions (as we will be learning more about strings in a video later than Week 4.3).

We will next concentrate on other functions to help in shipment. First, we will check whether a book of given dimensions (width, depth and height) can fit into a box of given dimensions:

- `book_fits_in_box(box_w, box_d, box_h, book_w, book_d, book_h)`: Returns `True` if the book of the given integer dimensions can fit in the box of the given integer dimensions, and `False` otherwise. Note: The book could be rotated or flipped in order to fit in the box.

There are three boxes available for books: a small box of 10x10x2 cm (WxDxH), a medium box of 15x15x3 cm, and a large box of 20x20x4 cm. Given the dimensions of a book, we will try to find the smallest box size that the book can fit in:

- `get_smallest_box_for_book(book_w, book_d, book_h)`: Returns a string corresponding to the smallest box size ('small', 'medium' or 'large') in which the book of given integer dimensions can fit. If none of the box sizes will fit the book, then return the empty string.

Sometimes there will be multiple copies of the same book (each having the same dimensions), and the question arises as to how many copies of the book will fit into one box of a given size:

- `get_num_books_for_box(box_w, box_d, box_h, book_w, book_d, book_h)`: Returns (as an integer) the maximum number of copies of a book of given integer dimensions that can fit into a box of given integer dimensions. Note that we will not try to rotate the books for this function; their orientation will remain fixed. That is, if the book has width 5, height 5 and depth 2, then it will still have width 5, height 5 and depth 2 when placed into the box.

Finally, we will write a function that lets the user access the various features of the program.

- `main()`: A greeting message should be shown to the user, followed by a menu of options, each with a number preceding the option's name. (See the example below for the option names and numbers.) When the user enters a valid option, you should prompt them for the required input (e.g., ISBN number and checksum, or box and/or book dimensions), then call the appropriate function(s) and print out the appropriate result message (e.g., 'ISBN is valid.', 'Package does not fit in box.', etc.). We will assume the user will always enter a number between 1 and 4.

Examples (as executed in Thonny)

Note: You can assume we will not test your functions with ISBN numbers that begin with a 0.

Example 1:

```
>>> calculate_isbn_checksum_by_digits(8, 7, 1, 1, 0, 7, 5, 5, 9)
7
```

Example 2:

```
>>> calculate_isbn_checksum(871107559)
7
```

Example 3:

```
>>> is_isbn(871107559, "4")
False
```

Example 4:

```
>>> book_fits_in_box(15, 2, 2, 2, 15, 2)
True

>>> book_fits_in_box(10, 10, 10, 2, 15, 2)
False
```

Example 5:

```
>>> get_smallest_box_for_book(12, 12, 2)
'medium'
```

Example 6:

```
>>> get_num_books_for_box(10, 5, 5, 5, 5, 2)
4
```

Example 7:

```
>>> main()
Welcome to the shipment calculation system.
1) Check ISBN
2) Check box/book size
3) Get smallest box size for book
4) Get num equally-sized books per box
Enter choice (1-4): 1
Enter ISBN: 100101011
Enter checksum: 1
ISBN is not valid (checksum did not match).
```

Question 3: Cards (30 points)

A deck of playing cards consists of 52 cards, each being a combination of a suit and rank. There are four possible suits (hearts, diamonds, clubs and spades) and thirteen possible ranks (the numbers from two to ten, and Jack, Queen, King and Ace). We will write a handful of Python functions that deal with playing cards represented as integers from 1 to 52.

Each integer from 1 to 52 will represent a different card of the deck in the following manner. We can take the numbers 1, 2, 3 and 4 to be Two's of each suit; the numbers 5, 6, 7, 8 as Three's of each suit, and so on, culminating in the numbers 49, 50, 51, 52 as Ace's of each suit. We will give the suits the order of Hearts, Diamonds, Clubs, and Spades; that is, the numbers 1, 2, 3, 4 will correspond to the Two of Hearts, Two of Diamonds, Two of Clubs and Two of Spades, respectively.

Thus, given any integer representation of a card (from 1 to 52), we can retrieve its rank and suit. For example, given the card 49, we can determine that it is suit 1 (Diamonds) and rank 12 (Ace). There will be a number given to each suit (between 0 and 3) and rank (between 0 and 12).

You will write your code for this question in a file called `card.py`. To begin, first define the following global variables at the top of your file. Note that we use all capitals for these global variables as they are 'constants': variables whose values will never change throughout the program's execution (rank two will always correspond to a 0, e.g.).

- `CLUBS, DIAMONDS, HEARTS, SPADES` (to be the numbers from 0 to 3, respectively)
- `ACE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING` (to be the numbers from 0 to 12, respectively)

Then, implement the following functions.

- `get_suit(card)`: Given an integer representation of a card, return its suit as an integer between 0 and 3.
- `get_rank(card)`: Given an integer representation of a card, return its rank as an integer between 0 and 12.
- `same_rank(card1, card2)`: Given two integer card representations, return `True` if they are both of the same rank, and `False` otherwise.
- `same_suit(card1, card2)`: Given two integer card representations, return `True` if they are both of the same suit, and `False` otherwise.
- `same_color_suit(card1, card2)`: Given two integer card representations, return `True` if they are both of the same color of suit, and `False` otherwise. Note that hearts and diamonds are red color suits, and clubs and spades are black color suits.

Note: If any of these functions use a long chain of conditional statements, marks will be deducted for style.

Examples (as executed in Thonny)

Example 1:

```
>>> %Run card.py
>>> get_suit(7)
3
```

Example 2:

```
>>> %Run card.py
>>> get_rank(7)
1
```

Example 3:

```
>>> %Run card.py
>>> same_rank(1, 3)
True
```

Example 4:

```
>>> %Run card.py
>>> same_suit(7, 15)
True
```

Example 5:

```
>>> %Run card.py
>>> same_color_suit(5, 6)
True
```

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.

`smoothie.py`
`shipment.py`
`card.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.