CS10 Python Programming Homework 3
20 points (Functions and Strings)
(no global variables allowed)
(No lists, tuples, sets, dicts allowed for this homework)

1. You must turn in your program listing and output for each program set. Each program set must have your student name, student ID, and program set number/description. Late homework will not be accepted for whatever reasons you may have.

*********_**for this homework, you are to submit your Program sets to Canvas**_ under Homework 3 link*********

   a. Name your files : HW3_PS1_ lastname_firstinitial.py for Program Set 1 and HW3_PS2_ lastname_firstinitial.py for PS2 and so on. PS means program set. If there are two program sets you will submit two files one for each program set. Example if your name is Joe Smith then the filename will be HW3_PS1_smith_j.py
   b. You must submit your homework by the deadline at the specified upload link on Canvas under homework 3. If the deadline is past, Program Sets will not be graded. Homework submitted via email attachment, comment in Canvas, Canvas message, or by any other method is not accepted and will be given a zero for no submission.
   c. if you do not follow the instructions on file naming provided in this section you will receive a zero for that program set that you did not correctly name the file.
   d. It is your responsibility to check if your homework is properly submitted to Canvas.

2. Please format your output properly, for example all dollar amounts should be printed with 2 decimal places when specified. Make sure that your output values are correct (check the calculations).

3. Use only the 'tools' in the topics we covered from lesson 1 to lesson 40 only. **No lists, tuples, sets, dictionary, classes. Global variables, break and continue are not allowed for this homework 3.**

4. Each student is expected to do their own work. **IF IDENTICAL PROGRAMS ARE SUBMITTED, EACH IDENTICAL PROGRAM WILL RECEIVE A SCORE OF ZERO.**

Grading:

Each program set must run correctly both syntactically, logically, and display the correct output exactly as specified. **If the program set does not run a zero will be given. If the program runs but does not display the correct output exactly as specified with proper formatting shown in the sample test run, a zero will be given**. If the program executes properly with proper syntax, logic, and displays the correct output with proper formatting as specified in the program set, you will receive the full points for that question. Then points will be deducted for not having proper:

a. Comments (1 pt. deducted for each occurrence)
   - Your name, description at the beginning of each program set. Short description of the what each section of your codes does.

b. Consistency/Readability (2 pts deducted for each occurrence)
   - Spacing(separate each section of codes with a blank line) but separate each function with 2 blank lines
   - Indentation
   - Style (proper naming of variables no a, b, c – use descriptive and mnemonics)
   - function **must include type hints or annotations** except for the main()
   - include **docstrings** for every function except for the main()

c. Required elements
   - Use only 'tools' in the topics that have been covered in class. For example, in for this homework you are only allowed to use the topics covered in class up to functions and strings(lesson 1 to lesson 40) only. Using **global variables and tools not in lesson 1 to lesson 40 like list, sets, dictionaries, tuple, class, break, continue will result in zero score for that program set.**

Grading continued…

d. Output (you **must provide the specified number of test runs or your program set will receive a zero score**)
   - to be displayed at the end of the program listing(codes) and commented
   - if no output(test runs) is provided for your uploaded file, a zero will be given for that program set.
   - must use test runs data when provided in the Program set question. Provide your own test runs if the program set does not ask for any.

Points will be deducted from grading items a. to d. above until your Program Set reaches zero points.

*******for all program sets you must use this statement in your program******

```
if __name__==”__main__”:
    main()
```

For all your functions you must include type annotations(type hints) and docstrings

Example of type annotation for multiple return values and docstring in triple quotes.

```
def function(var1 : int, var2 : int, var3 : int) -> (str, float):
    '''Some codes that do something'''          #this is a docstring
    pass                                        # a function can exist with no codes if you include pass as a line of code
    return strvar1, floatvar2                   # this line is just to demonstrate the return values, the return statement is
                                                #not required if you have "pass". Pass allows a function to exists without
                                                #codes inside the function, so that you can fill in the codes later.
```

Program Set 1 Using Functions (10 points)

Global variables not allowed for this program set. If you use global variable a zero will be given for this program set 1. See above item 3 and grading item c.

Stock Transaction Program (**same program as in HW1 except you must convert the program to using functions. Also see lesson 33 the example is similar to PS1**)

Last month Joe purchased some stock from StockTrade.

1. **Write a function(s) to allow the user to input the followings**:

   - The name of the stock
   - Number of shares Joe bought
   - Stock purchase price
   - Stock selling price
   - Broker commission

2. **Write a function(s) to calculate**:
   - The amount of money Joe paid for the stock (number of shares bought * purchase price)
   - The amount of commission Joe paid his broker when he bought the stock. (Amount he paid for stocks * commission in percent)
   - The amount that Jim sold the stock for. (number of shares * selling price)
   - The amount of commission Joe paid his broker when he sold the stock. (Amount he sold shares * commission in percent)
   - Calculate the amount of money Joe had left when he sold the stock and paid his broker (both times). If this amount is positive, then Joe made a profit. If the amount is negative, then Jim lost money.
   - Profit/loss =(amount for sold stocks- commission) - (amount paid to buy stocks + commission)

3. **Write a function(s) to display the following paid for the stock(s) that Joe had transacted (if Joe entered 5 sets of stocks transactions then output 5 sets of stocks).**

- The Name of the Stock
- The amount of money Joe paid for the stock (number of shares bought * purchase price)
- The amount of commission Joe paid his broker when he bought the stock. (Amount he paid for stocks * commission in percent)
- The amount that Jim sold the stock for. (number of shares * selling price)
- The amount of commission Joe paid his broker when he sold the stock. (Amount he sold shares * commission in percent)
- Display the amount of money Joe had left when he sold the stock and paid his broker (both times). If this amount is positive, then Joe made a profit. If the amount is negative, then Jim lost money. Profit/loss =(amount for sold stocks- commission) - (amount paid to buy stocks + commission)

Write the main() function to call the above functions.

You must have at least 4 functions, a function for all the user inputs, a function for calculations, a function for output and the main function. You can have more functions other than the 4 functions described.

You should also allow the user to enter as many stocks as he/she wants or not to enter any stock at all with 'Yes' or 'No' (use the proper loop). (this PS have 5 test runs) Your output must look exactly like the specifications given below if you do not correctly display the output you will get a zero. If you have done the formatting correctly in homework 1 then use the same formatting.

Assume that the user will always enter the Stock Name as a string and not anything else. The output amount for amount paid and amount sold will not exceed 9,999,999.99

10 blank spaces — 13 columns → use format( ) to specify the column width and decimal places.

```
Amount paid for the stock:          $    339,200.00
Commission paid on the purchase:    $     13,568.00
Amount the stock sold for:          $    359,200.00
Commission paid on the sale:        $     14,368.00
Profit (or loss if negative):       $     -7,936.00
```

Here are 2 sample runs(blue user input):

```
##Test Run 1

##Enter your stock information? Type 'y' for yes, or 'n' for no: y
##
##Enter Stock name: Kaplack, Inc.
##Enter Number of shares : 10000
##Enter Purchase price : 33.92
##Enter selling price : 35.92
##Enter Commission : 0.04
##
##
##Stock name : Kaplack, Inc.
##Amount paid for the stock:          $    339,200.00
##Commission paid on the purchase:    $     13,568.00
##Amount the stock sold for:          $    359,200.00
##Commission paid on the sale:        $     14,368.00
##Profit (or loss if negative):       $     -7,936.00


##Enter your stock information? Type 'y' for yes, or 'n' for no: y
##
##Enter Stock name: IBM
```

```
##Enter Number of shares : 15000
##Enter Purchase price : 50.25
##Enter selling price : 100.20
##Enter Commission : 0.02
##
##
##Stock name : IBM
##Amount paid for the stock:          $   753,750.00
##Commission paid on the purchase:    $    15,075.00
##Amount the stock sold for:          $ 1,503,000.00
##Commission paid on the sale:        $    30,060.00
##Profit (or loss if negative):       $   704,115.00

##Enter your stock information? Type 'y' for yes, or 'n' for no: n
>>>

## Test Run 2 (test the loop with n to exit program)
## Enter your stock information? Type 'y' for yes, or 'n' for no: n
## >>>
```

You will provide test Run 3, 4 and 5 using you own data. Each test run must have at least 2 to 3 stocks entered with your own data.

Continue next page for Program Set 2…

Program Set 2 (10 points)

Global variables not allowed for this program set. If you use global variable a zero will be given for this program set 2. See above item 3 and grading item c.

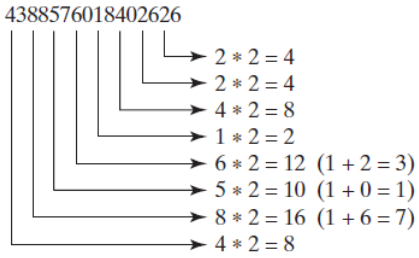
Credit card number validation Program

Credit card numbers follow certain patterns: It must have between 13 and 16 digits, and the number must start with:

- 4 for Visa cards
- 5 for MasterCard credit cards
- 37 for American Express cards
- 6 for Discover cards

In 1954, Hans Luhn of IBM proposed an algorithm for validating credit card numbers. The algorithm is useful to determine whether a card number is entered correctly or whether a credit card is scanned correctly by a scanner. Credit card numbers are generated following this validity check, commonly known as the *Luhn check* or the *Mod 10 check*, which can be described as follows (for illustration, consider the card number 4388576018402626):

1. Double every second digit from right to left. If doubling of a digit results in a two-digit number, add up the two digits to get a single-digit number.



2. Now add all single-digit numbers from Step 1.

   4 + 4 + 8 + 2 + 3 + 1 + 7 + 8 = 37

3. Add all digits in the odd places from right to left in the card number.

   6 + 6 + 0 + 8 + 0 + 7 + 8 + 3 = 38

4. Sum the results from Steps 2 and 3.

   37 + 38 = 75

5. If the result from Step 4 is divisible by 10, the card number is valid; otherwise, it is invalid. For example, the number 4388576018402626 is invalid, but the number 4388576018410707 is valid.

Write a program that prompts the user to enter a credit card number as a **STRING**. **Allow the user enter multiple credit numbers to test for validity for each test case run(use a for loop)**. Prompt the user for how many credit card numbers he/she wants to check. **You are to use the two credit card numbers provided in the sample output given below plus other credit card numbers using credit numbers that starts with 5, 37, and 6. Make sure you test at least 10 credit card numbers.**

Display whether the number is valid or invalid. Design your program to use the following functions shown below (include type annotations and docstrings, **you must only use the functions template provided. You are *not allowed to add extra functions, remove or change the name of any of the functions that are provided. Any changes made to the provided functions template below will result in a zero for the whole program set. You are allowed to remove the comments below each of the function headers in green fonts)**:

Functions Template:

```
def isValid(number):
    # Return true if the card number is valid
    # hint you will have to call function sumOfDoubleEvenPlace() and sumOfOddPlace

def sumOfDoubleEvenPlace(cardNumber):
    # Get the result from Step 2

def getDigit(number):
    # Return this number if it is a single digit, otherwise, return
    # the sum of the two digits

def sumOfOddPlace(cardNumber):
    # Return sum of odd place digits in number

def main():
    #user will input the credit card number as a string
    #call the function isValid() and print whether the credit card number is valid or not valid
```

Here are the three sample runs with credit card numbers you have to use for test runs 1,2,3:

```
## Test Run 1

## How many credit card numbers do you want to check? 2
## Enter a credit card number: 4388576018402626
## 4388576018402626 is invalid
##
## Enter a credit card number: 4388576018410707
## 4388576018410707 is valid
## >>>

##
## Test Run 2
## How many credit card numbers do you want to check? 0
## >>>

##
## Test Run 3
## How many credit card numbers do you want to check? 3
## Enter a credit card number: 12345678
## 12345678 is invalid
##
## Enter a credit card number: 5169769005222217
## 5169769005222217 is valid
##
## Enter a credit card number: 6011655276746808
## 6011655276746808 is invalid
## >>>
```

Test run 4
…

Test run 5
…

>>>
Provide the last 2 test runs which must also test credit cards starting with 5, 37, and 6, testing for valid and not valid for each type of credit card type.
>>>

Some notes to help with Program Set 2

Searching for Substrings

You can search for a substring in a string by using the methods below:

str	
endswith(s1: str): bool	Returns True if the string ends with the substring s1.
startswith(s1: str): bool	Returns True if the string starts with the substring s1.
find(s1): int	Returns the lowest index where s1 starts in this string, or –1 if s1 is not found in this string.
rfind(s1): int	Returns the highest index where s1 starts in this string, or –1 if s1 is not found in this string.
count(substring): int	Returns the number of non-overlapping occurrences of this substring.

Example:

```
 1 >>> s = "welcome to python"
 2 >>> s.endswith("thon")
 3 True
 4 >>> s.startswith("good")
 5 False
 6 >>> s.find("come")
 7 3
 8 >>> s.find("become")
 9 -1
10 >>> s.rfind("o")
11 17
12 >>> s.count("o")
13 3
14 >>>
```

Since **come** is found in string **s** at index **3**, **s.find("come")** returns **3** (line 7). Because the first occurrence of substring **o** from the right is at index 17, **s.rfind("o")** returns **17** (line 11). In line 8, **s.find("become")** returns **-1**, since **become** is not in **s**. In line 12, **s.count("o")** returns **3**, because **o** appears three times in **s**.

Here is another example:

```
s = input("Enter a string: ")
if s.startswith("comp"):
print(s, "begins with comp")
if s.endswith("er"):
print(s, "ends with er")
print('e', "appears", s.count('e'), "time in", s)
```

```
>>>
If you enter computer when running the code, it displays
computer begins with comp
computer ends with er
e appears 1 time in computer
>>>
```