CS10 Python Programming Homework 3
40 points
Functions and Strings

(No lists, tuples, sets, dicts allowed for this homework except for what already covered in lectures)

1. You must turn in your program listing and output for each program set. Start a new sheet or sheets of paper for each program set. Each program set must have your student name, student ID, and program set number/description. All the 3 program sets must be printed and submitted together with your Exam 3. Once you complete your Exam 3 and leave the classroom you will not be able to submit Homework 3. Late homework will not be accepted for whatever reasons you may have.

****

for this homework, you are also to submit All Program Sets to Canvas under Homework 3 link

****

a. Name your file : PS1_firstinitial_lastname.py for Program Set 1. PS means program set
b. You still have to submit the paper copy together with the rest of the Homework 3.
c. You have till 11:59pm on the night before Exam 3 to submit all Program Sets to Canvas. If the deadline is past, Program Sets will not be graded even if you submit the paper copy on time.
d. You must submit both hardcopy and upload the Program Sets files to Canvas to be graded. If you only submit the hardcopy or only upload the file to Canvas you will receive a zero for that Program Set. You must submit all the hardcopies of all 3 Program Sets of Homework 3 and upload the files of all 3 program sets to Canvas.
e. if you do the extra credit you must also submit hardcopy and upload to Canvas. Name the file EC_firstinitial_lastname.py
f. if you do not follow instructions on file naming provided in this section you will receive a zero for the whole of Homework 3.

2. You must STAPLE (not stapled assignments will not be graded resulting in a zero score) your programming assignment and collate them accordingly. Example Program set 1 listing and then output, followed by Program Set 2 listing and output and so on.

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

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 ***syntactically, logically, and display the correct output as specified***. If the program set does not run correctly, a zero will be given. For each Program set, if the program executes properly with proper syntax, logic, and displays the correct output, 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 do.
b. Consistency/Readability (2 pts deducted for each occurrence)
   - Spacing(separate each section of codes with a blank line
   - Indentation
   - Style (proper naming of variables no a,b,c – use descriptive and mnemonics)
   -each function must include type hints or annotations except for the main()
   -include docstrings for every function
c. Required elements (2 pts deducted for each occurrence)
   - Use tools that have been covered in class
   - proper formatting for output when specified
   - all monetary values must be in 2 decimal places
d. Output

- if no output is provided for either the hardcopies or uploaded file, a zero will be given for that program set.
- Output must to be displayed at the end of the program listing(codes)
- must use test cases when provided in the Program set question. Provide your own test cases if the program set does not ask for any. The minimum test cases you provide on your own is 5 or more. If you provide less then 5 test cases per Program Set then that program set will receive a zero grade.

*******for all program sets you must use this statement in your program******
```
if __name__=="__main__":
```

Program 1 Using Functions (10 points)

Stock Transaction Program

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 function(s) to calculate: and:
   - 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 paid to buy stocks + commission) – (amount for sold stocks– commission)

3. Write 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 paid to buy stocks + commission) – (amount for sold stocks– commission)

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

For all your functions you must include type annotations. **You should also allow the user to enter as many stocks as he/she wants or not to enter any stock at all**. (for this hw have at least 5 test cases)

Example of type annotation for multiple return values;

```
def function(var1:int, var2:int, var3:int)->( str, float)
        '''Some codes that do something'''
        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,
                                # not required if you have pass above.
```

Program 2 Strings(10 points)

Write a program to check whether a string entered is a palindrome. A string is a palindrome if it reads the same forward and backwards. The words “mom, ‘dad”, “noon” for instance are palindromes. (include type annotations for all functions except for main, the python ***reverse()*** method is **not allowed** to be used with this program)

1. write a function def isPalindrome() to check if the string entered is a palindrome.
2. write the main() function to allow the user to enter the string and prints if it is a palindrome or not.
3. Use at least 5 test cases 2 for true and 2 for false and one test case to quit out before entering the loop.
4. Allow the user to check as many words for each run of the program.

Output sample:

Enter a String : noon<ENTER>
noon is a palindrome

Enter a String : print<ENTER>
print is not a palindrome

Enter a String : racecar<ENTER>
racecar is a palindrome

Enter a String : speed<ENTER>
speed is not a palindrome

Enter a String : -1<ENTER>
>>>Program quits

Program Set 3 (20 points)

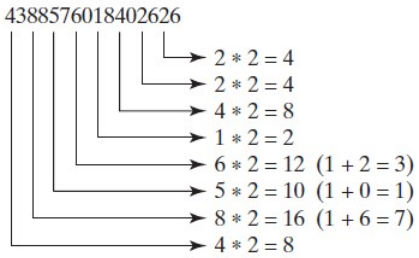
Credit card number validation Program

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 to **check multiple credit numbers until user decides to quit**. 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 (include type annotations):

```
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
```

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

def sumOfDoubleEvenPlace(number):
    # 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(number):
    # Return sum of odd place digits in number
```

Here are the two sample runs using the credit card numbers above:

```
>>>
============ RESTART: E:/ /Homework/HW3/HWP_3CC.py ============
Enter a credit card number as a long integer: 4388576018402626
4388576018402626 is invalid

Enter a credit card number as a long integer: 4388576018410707
4388576018410707 is valid
```

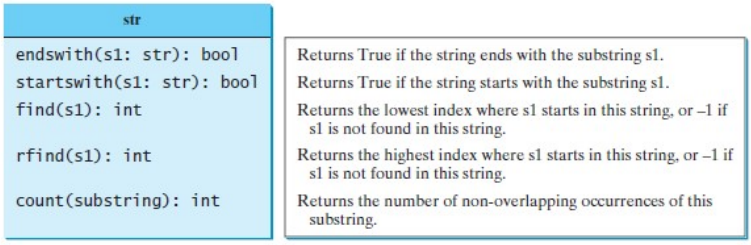
Plus 8 test cases more with credit cards starting with 5, 37, and 6, testing for valid and not valid for each type of credit card type.

```
>>>
```

Some notes to help with Program Set 3

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:

```
>>> s = "welcome to python"
>>> s.endswith("thon")
True
>>> s.startswith("good")
False
>>> s.find("come")
3
>>> s.find("become")
-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
>>>
```

Program Set 4(10 points extra credit)

Biologists use a sequence of letters **A**, **C**, **T**, and **G** to model a *genome*. A *gene* is a substring of a genome that starts after a triplet **ATG** and ends before a triplet **TAG**, **TAA**, or **TGA**. Furthermore, the length of a gene string is a multiple of **3** and the gene does not contain any of the triplets **ATG**, **TAG**, **TAA**, and **TGA**.

Write a program that prompts the user to enter a genome and displays all genes in the genome. If no gene is found in the input sequence, the program displays **no gene is found**. **Allow the user to repeat the process as many times as the user wants until the user decides to quit**. Here are the sample runs:

```
>>>
========== RESTART: E:/HW3/HW3_4_genes.py ==========
Enter a genome string: TTATGTTTTAAGGATGGGGCGTTAGTT
TTT
GGGCGT

Enter a genome string: TGTGTGTATAT
no gene is found
>>>
```

Test with 4 more genome strings.
TGATGCTCTAAGGATGCGCCGTTGATT
TGATGCTCTAGAGATGCGCCGTTGAATAT
TGATGCGTCTAAGAGACTGCTCGCCGGTTGAATAT
TGATGGCTCCTATGAGAATGGCGCCCGTTTCGAAATAT