Lab 3 - Strings and Loops

Q1: Input Validation

We often need to use while-loops to validate a user's input. This exercise is all about validating user inputs.

a) **[*]** Prompt the user for a positive integer. Keep prompting the user until the integer entered by the user is indeed positive. (You can assume that the user always enters an integer, but the user may enter a negative integer or 0, which is not considered valid.)

   See a sample run below. Texts in bold are user inputs.

```
Enter a positive integer: -2
Error! Enter a positive integer: 0
Error! Enter a positive integer: -5
Error! Enter a positive integer: 4
Bingo!
```

b) **[*]** Prompt the user for a single word. Keep prompting the user until the string entered by the user doesn't contain any space.

   See a sample run below. Texts in bold are user inputs.

```
Enter a single word: Python Programming
Error! Enter a single word: New York City
Error! Enter a single word: SMU
Bingo!
```

c) **[*]** Prompt the user for his/her age. Keep prompting the user until the input is between 0 and 100.

```
How old are you? -3
Error! How old are you? 101
Error! How old are you? 99
Thanks!
```

d) **[*]** Ask the user whether he/she is a student. If the answer from the user is neither "yes" nor "no", keep prompting the user until the answer is valid. The program should be flexible enough to accept "YES", "Yes" and "yes" (but not "yEs", "YEs", etc.). Similarly for "no".

   A sample run is below. Texts in bold font are user inputs.

```
Are you a student? I am
Invalid answer! Are you a student? maybe
Invalid answer! Are you a student? No
```

```
Thanks!
```

Another sample run is below.

```
Are you a student? I don't know.
Invalid answer! Are you a student? Can you guess?
Invalid answer! Are you a student? YES
Thanks!
```

Q2: Numbers (Part 1)

[*] Prompt the user for a sequence of numbers. Keep prompting until the sum of all the numbers have exceeded 100. A sample run can be found below:

```
Enter a number: 10
Enter a number: 30
Enter a number: 20
Enter a number: 35
Enter a number: 10
Thanks! The sum is 105.
```

Q3: Voucher

[*] You have a $100 voucher to spend in a bookstore. Because the voucher has to be spent on a single receipt, you need to buy at least $100 of books. Write a program that helps you check whether the books you've picked are enough to use up the voucher. The program should keep prompting you to buy one more book until the total price is equal to or greater than $100. A sample run of the program can be found below:

```
Please pick a book. How much is it? $ 10
You still have $90.0 left.
Please pick a book. How much is it? $ 15.5
You still have $74.5 left.
Please pick a book. How much is it? $ 53.5
You still have $21.0 left.
Please pick a book. How much is it? $ 10
You still have $11.0 left.
Please pick a book. How much is it? $ 12
The total price is $101.0.
```

Q4: Numbers (Part 2)

[**] Prompt the user for a sequence of integers until the user enters a negative integer. The program then displays how many even numbers have been entered, excluding the last negative integer.

A sample run of the code can be found below:

```
Enter an integer: 7
Enter an integer: 3
Enter an integer: 8
Enter an integer: 12
Enter an integer: 23
Enter an integer: 0
Enter an integer: 3
Enter an integer: -9
You've entered 3 even numbers (excluding the last negative number).
```

Q5: Print a Message with Separators

Part I [*]

Prompt the user for a message. Display the message character by character but **separate every two adjacent characters by a space**.

Your code should produce the following output:

```
Enter a message :Hello!
H e l l o !
```

```
Enter a message :Python
P y t h o n
```

Hint:

- Use a for-loop to go through all the characters in the message one by one.
- Use `print(s, end='')` to print out `s` and still keep the cursor in the same row.

Part II [**]

Now write another piece of code that displays the message from the user character by character but separate every two adjacent characters **by a hyphen**.

Your code should produce the following output:

```
Enter a message :Hello!
H-e-l-l-o-!
```

```
Enter a message :Python is fun!
P-y-t-h-o-n- -i-s- -f-u-n-!
```

Hint: The last character should be handled differently from the other characters.

Part III [**]

Now let us make the code above re-usable and more general.

Define a function called `print_message_with_separators` that does the following:

- The function takes in two parameters:
  - `msg` (of type `str`): a message string (e.g., `'Hello!'`)
  - `sep` (of type `str`): a separator string (e.g., `'-'`)

- The function **prints out** `msg` character by character, but it inserts `sep` between every two adjacent characters of `msg`.
- The function **does not return any value**.

For example, when `print_message_with_separators("Hello World!", '*')` is called, we should see the following output:

```
H*e*l*l*o* *W*o*r*l*d*!
```

We can see that the separator `'*'` is inserted everywhere between two adjacent characters in the message.

When `print_message_with_separators("Python", '/')` is called, we should see the following output:

```
P/y/t/h/o/n
```

Q6: Display Numbers [**]

Define a function called `display_numbers`. The function takes in two parameters: **`m`** and **`n`**. It is assumed that both **`m`** and **`n`** are integers, and **`m`** is less than or equal to **`n`**. The function displays all the integers between **`m`** and **`n`** (both inclusive) one by one, separated by spaces. However, the function handles the following numbers in a special way:

- If the number is a multiple of 3 but not a multiple of 5, the function displays a hyphen instead of the number.
- If the number is a multiple of 5 but not a multiple of 3, the function displays an asterisk instead of the number.
- If the number is a multiple of both 3 and 5, the function displays a # instead of the number.

For example, calling `display_numbers(4, 16)` gives the following output to the screen:

```
4 * - 7 8 - * 11 - 13 14 # 16
```

Q7: Nested if/else [*]

What values of **a**, **b**, **c** and **d** can cause the code below to display the following output?

- `Alpha`
- `Beta`
- `Gamma`
- `Delta`

```
if a and b:
    if c:
        print('Alpha')
    else:
        print('Beta')
else:
    if d:
        print('Gamma')
    else:
        print('Delta')
```

output	value of `a`	value of `b`	value of `c`	value of `d`
`Alpha`				
`Beta`				
`Gamma`				
`Delta`				

Q8: Evaluation of Expressions [**]

What are the values of the following expressions?

- `not True or (3 >= 3 or 9 < 4) and False`
- `not True or 3 >= 3 or 9 < 4 and False`
- `False == (False or not True) or not (2 * 4 % 3 == 1)`

You may want to check the Operator Precedence table provided below to evaluate the expressions.

Operator	Description
**	Exponent
*, /, %, //	Multiplication, division, modulo and floor division
+, -	Addition and subtraction
==, !=, <, <=, >, >=	Relational Operators
not	Boolean NOT
and	Boolean AND
or	Boolean OR

Q9: De Morgan's Law [**]

Simplify the following expression.

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

Simplified expression: ____.

Now check your answer by plugging in all combinations of possible values of `a` and `b` and evaluating the expression above as well as your simplified expression. Fill in your evaluations in the table below.

a	**b**	**not (not (a and not b) or (b or not a))**	**Your simplified expression**
True	True		
True	False		
False	True		
False	False		

Q10: Compute Sum [**]

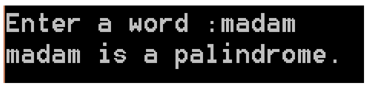
Write a function called `compute_sum` that takes in two parameters, **m** and **n**. The function returns the sum of all the integers between **m** and **n**.

Then use your function. For example, if you call the function with arguments 4 and 10, the function should print 49.

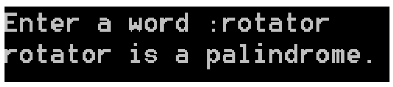
Q11: Palindrome [***]

A word is called a palindrome if it reads the same backwards as forwards. For example, "**madam**" and "**rotator**" are examples of palindromes.

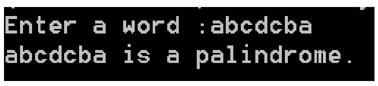
Write a program that prompts the user for a word and checks whether the word is a palindrome. Your code should produce the following output:



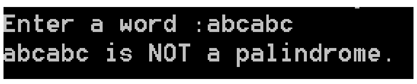
```
Enter a word :madam
madam is a palindrome.
```



```
Enter a word :rotator
rotator is a palindrome.
```



```
Enter a word :abcdcba
abcdcba is a palindrome.
```



```
Enter a word :abcabc
abcabc is NOT a palindrome.
```

Q12: Buying Books [***]

Write a simple program that helps a bookstore's customer to gather some statistics of the books she has picked.

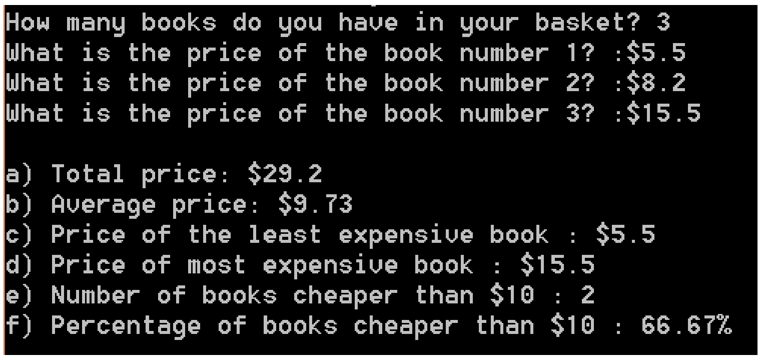
The program first asks the customer how many books she has in her basket. The program then prompts the customer for the price of each book. After that, the program does the following:

a) The program displays the total price of all the books.

b) The program displays the average price of all the books.
c) The program displays the price of the least expensive book
d) The program displays the price of the most expensive book.
e) The program displays how many books are cheaper than $10.00.
f) The program displays the percentage of books cheaper than $10.00.

Note: You may want to use the `round()` function to limit the number of digits after the decimal point to 2. For example, `round(x,2)` will round x to 2 decimal places.

Your code should produce the following output:



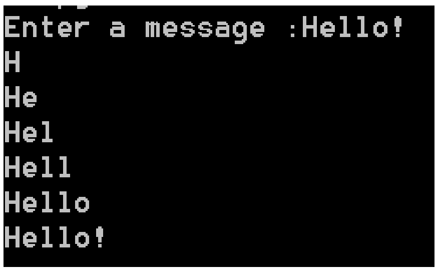
```
How many books do you have in your basket? 3
What is the price of the book number 1? :$5.5
What is the price of the book number 2? :$8.2
What is the price of the book number 3? :$15.5

a) Total price: $29.2
b) Average price: $9.73
c) Price of the least expensive book : $5.5
d) Price of most expensive book : $15.5
e) Number of books cheaper than $10 : 2
f) Percentage of books cheaper than $10 : 66.67%
```

Q13: Playing with Strings [***]

Part I

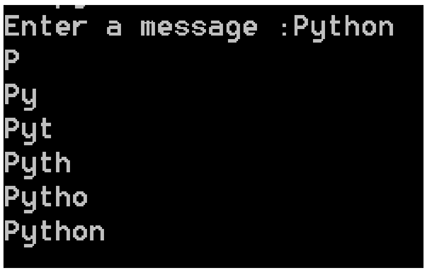
Write a program that prompts the user for a string as a message. The program displays a triangle that gradually reveals the whole message, as shown below.

Suppose the message is “Hello!”. The program displays the following output:



```
Enter a message :Hello!
H
He
Hel
Hell
Hello
Hello!
```

Suppose the message is “Python”. The program displays the following output:

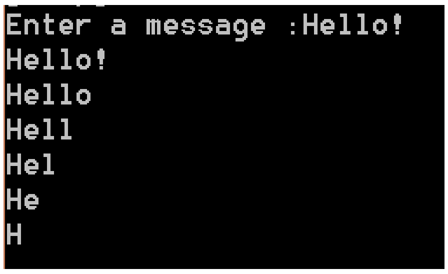


```
Enter a message :Python
P
Py
Pyt
Pyth
Pytho
Python
```

Part II

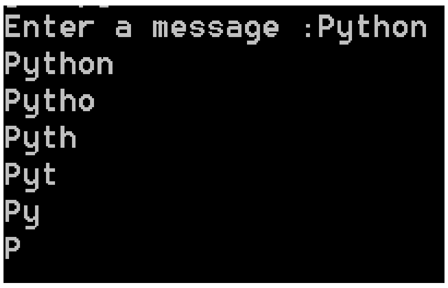
Write a similar program that displays the triangle upside down, as shown below.

Suppose the message is “Hello!”. The program displays the following output:



```
Enter a message :Hello!
Hello!
Hello
Hell
Hel
He
H
```

Suppose the message is “Python”. The program displays the following output:



```
Enter a message :Python
Python
Pytho
Pyth
Pyt
Py
P
```

Q14: Count Words [**]

In this question we’ll count the number of “a” and “an” in a piece of text.

Note: For this question, you are NOT allowed to use the `count()` method of string.

a) Write a function called `count_a` that takes in a piece of text (as a string) and returns the number of times the word “a” occurs in the text. You do not need to handle uppercase “A”. You can assume that each time the word “a” occurs, both its previous character and its next character are a space.

   For example, `count_a("I have a room with a window, a desk and a chair.")` should return `4`.

b) Write a function called `count_an` that takes in a piece of text (as a string) and returns the number of times the word “an” occurs in the text. You do not need to handle uppercase “An”. You can assume that each time the word “an” occurs, its previous character and its next character are both a space.

   For example, `count_an("Every day I have an egg, an apple and a banana for breakfast.")` should return `2`.

Q15: Fibonacci Numbers [***]

Refer to the following link to understand Fibonacci numbers:

https://en.wikipedia.org/wiki/Fibonacci_number

Essentially, it is a sequence of numbers where each number is equal to the sum of its previous two numbers in the sequence.

Write a function called `display_fibonacci()`. This function takes in an integer `n` (greater or equal to 3). It **prints out** the first `n` Fibonacci numbers, starting from 1. The function doesn't return anything.

For example:

`display_fibonacci(3)` prints out the following output: `1 1 2`

`display_fibonacci(5)` prints out the following output: `1 1 2 3 5`

`display_fibonacci(10)` prints out the following output: `1 1 2 3 5 8 13 21 34 55`