**Lab 3: About Strings**

Learning Outcomes:

* Understand how to use string methods by passing appropriate values to the methods
* Get to solve problems involving string objects

Instructions:

* Suggest that you create a working folder **is111\lab3** in your **C** or **D** drive. Store all your solutions in this working folder.
* Challenging questions are marked with \*.

To submit:

* Please submit your working solutions via your assignment Dropbox in eLearn **within 1 week**. The Dropbox will be closed after the due date.
* Zip up all your source files into a single zip file called **<your email ID>\_lab3.zip** (e.g. **ahlian.lim.2011\_lab3.zip**). You should only submit a single zip file for each lab.

1. Write a program in a file named **lab3\_1.py** that prompts a user for his/her name and gender and writes it back with the surname in capital letters and the rest of the name with first letter in uppercase. The program should also add the prefix Mr. or Ms. to the name based on the user input for gender. Assume the following:

* The given name could be one word, or more
* If the name has more than 1 word, the first word is always the surname
* Two adjacent words of the name is separated by 1 space

Note: Take into consideration all possibilities of names before you start writing your code. Make use of **if-else** effectively i.e. avoid redundant use of **if-else** structure.

Look for String methods in Python documentation [https://docs.python.org/3/tutorial/introduction.html#strings](https://docs.python.org/3/tutorial/introduction.html%23strings) to help you write this program. https://docs.python.org/3/library/stdtypes.html#string-methods

Below are two sample runs of the program:

|  |
| --- |
| D:\is111\lab3>**python lab3\_1**  Enter your name :**tan wee kiat**  Enter M or F for gender :**M**  Hello Mr. TAN Wee Kiat |

|  |
| --- |
| D:\is111\lab3>**python lab3\_1**  Enter your name :**hazirah**  Enter M or F for gender :**F**  Hello Ms. Hazirah |

To submit**: lab3\_1.py**

1. Write a program in a file named **lab3\_2.py** that prompts the user for a line of input containing an email address. The program then displays the *first* email address contained in that line of input.

**Assumptions:**

* + An email address embedded in an input line will be surrounded by an empty space at both ends.
  + Each line of input will always contain at least one email address.

For example, given the input “Email: contact\_us@smu.edu.sg” as shown below, the program will extract “contact\_us@smu.edu.sg” and display it.

Below are three sample runs of the program:

|  |
| --- |
| D:\is111\lab3>**python lab3\_2.py**  Enter a line of text with email address :**Mail us at contact\_us@smu.edu.sg with your queries.**  Email address:contact\_us@smu.edu.sg |

|  |
| --- |
| D:\is111\lab3>**python lab3\_2.py**  Enter a line of text with email address :**Mail us at contact\_us@smu.edu.sg or xyz@smu.edu.sg with your queries.**  Email address:contact\_us@smu.edu.sg |

|  |
| --- |
| D:\is111\lab3>**python lab3\_2.py**  Enter a line of text with email address :**contact\_us@smu.edu.sg**  Email address:contact\_us@smu.edu.sg |

**Hint**:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| .. | .. | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |  |
|  |  | : |  | c | o | n | t | a | c | t | \_ | u | s | @ | s | m | u | . | e | d | u | . | s | g |  |

symbolIndex

startIndex

endIndex

In order to extract the first email address in the input line, you will need to locate the boundary of the email address by obtaining the index numbers of the startIndex and the endIndex. How can you find these two index numbers? If you think carefully, you will see that startIndex marks the last space before the symbol ‘@’ while endIndex marks the first space after the symbol ‘@’. Let us refer to the index of ‘@’ as symbolIndex. Look for functions in the String module to find indices of startIndex, endIndex and symbolIndex

A special scenario is when the email address is either at the very beginning or at the very end of the input line (or both). In this case, there may not be a space before (or after) the email address. To work around this, you can first concatenate a space to both ends of the input line. Thus, if the input line is “xyz@smu.edu.sg”, it becomes “ xyz@smu.edu.sg ”.

To submit**: lab3\_2.py**

1. Prompt the user to enter a string, say text, and two characters, say start and end. Write code in a file named **Lab3\_3.py** that searches for a substring in the string say text such that the substring begins with the character start and ends with the character end. If there is no such substring, display the message "No such substring". The text, start and end are string values input by the user.

Sample runs of the program should look like this:

|  |
| --- |
| D:\IS111\lab3>**python lab3\_3.py** Enter the string :**Fibonacci series**  Enter the start char :**F**  Enter the end char :**b**  Substring [Fib] is found  D:\IS111\Lab3>**python lab3\_3.py** Enter the string :**Fibonacci series**  Enter the start char :**b**  Enter the end char :**s**  Substring [bonacci s] is found  D:\IS111\Lab3>**python lab3\_3.py** Enter the string :**Fibonacci series**  Enter the start char :**b**  Enter the end char :**y**  No such substring |

To submit**: lab3\_3.py**

1. \*A palindrome is a string that reads the same forward or reverse. Prompt the user to enter a string and print the message “The string <*input string*> is a palindrome” if the input string is a palindrome, else print the message “The string <*input string*> is not a palindrome”

The program should ignore digits, empty spaces or any special characters in the input string when evaluating for a palindrome.

Note: Look for String methods in Python documentation <https://docs.python.org/3/library/stdtypes.html#string-methods> to help you write this program

Sample runs of the program should look like this:

|  |
| --- |
| D:\IS111\lab3>**python lab3\_4.py** Enter the string :**madam**  The string "madam" is a palindrome  D:\IS111\Lab3>**python lab3\_4.py** Enter the string :**No 'x' in Nixon**  The string "No 'x' in Nixon" is a palindrome  D:\IS111\Lab3>**python lab3\_3.py** Enter the string :**bad, chocolate dab**  The string "bad, chocolate dab" is not a palindrome |

To submit**: Lab3\_4.py**

1. \*Write a program in a file named **lab3\_5.py** that prompts the user for two strings. Let us call the first string str1 and the second string str2. The program displays "Bingo!" if every character in str1 also appears in str2. Otherwise, the program displays "Nope :( "

Some sample runs of the program are shown below:

|  |
| --- |
| D:\IS111\Lab3>**python lab3\_5.py** Enter characters: **daily**  In string? **Holiday**  Bingo!  D:\IS111\Lab3>**python lab3\_5.py**  Enter characters: **lily**  In string? **Holiday**  Bingo!  D:\IS111\Lab3>**python lab3\_5.py** Enter characters: **pokemon**  In string? **pogo**  Nope :( |

To submit**: lab3\_5.py**

1. \* To verify the accuracy and validity of credit card numbers, most credit card numbers are encoded with a "check digit". A check digit is a digit added to a number (at either the end or the beginning) that validates the validity of the number. A simple algorithm is applied to the other digits of the number, which yields the check digit.

The steps for validation are as follows:

* 1. The alternate digits from the left most digit or the digits in the even position starting from the right are multiplied by 2, and then reduced to a single digit by subtracting 9.
  2. Add the unaffected digits (shown in blue below) in the original number with individual digits comprising the products obtained in step (i).
  3. The total obtained in step (ii) must be divisible by 10.

For example, given the number 2323 2005 7766 3554, this is how step (i) is performed:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Multiply by 2 | Subtract 9 if value >= 10 |
| 1st digit | 2 | 2 \* 2 = 4 | 4 |
| 2nd digit | 3 |  |  |
| 3rd digit | 2 | 2 \* 2 = 4 | 4 |
| 4th digit | 3 |  |  |
| 5th digit | 2 | 2 \* 2 = 4 | 4 |
| 6th digit | 0 |  |  |
| 7th digit | 0 | 0 \* 2 = 0 | 0 |
| 8th digit | 5 |  |  |
| 9th digit | 7 | 7 \* 2 = 14 | 14 - 9 = 5 |
| 10th digit | 7 |  |  |
| 11th digit | 6 | 6 \* 2 = 12 | 12 - 9 = 3 |
| 12th digit | 6 |  |  |
| 13th digit | 3 | 3 \* 2 = 6 | 6 |
| 14th digit | 5 |  |  |
| 15th digit | 5 | 5 \* 2 = 10 | 10 - 9 = 1 |
| 16th digit | 4 |  |  |

In step (ii), the following numbers are added together: 4 + 3 + 4 + 3 + 4 + 0 + 0 + 5 + 5 + 7 + 3 + 6 + 6 + 5 + 1 + 4, which yields the answer 60. Because 60 is divisible by 10, this credit card number is a valid one.

Write a program in a file named **lab3\_6.py** that prompts the user to enter a 16-digit credit card number, and then prints a message to indicate if the credit card number is valid. Here are three valid credit card numbers for testing purposes:

4041422060806790

5222747000084993

5256392810443201

Hint: You could treat the card number as string to read alternate digits. for statement when used like this,  
 for i in range(0,5,2):  
 print(i)  
 prints the numbers 0 2 and 4 because it prints numbers from 0 until 5 with an incremental step of 2

To submit:**lab3\_6.py**