



GCS FINAL PROJECT

Students: Jana Hariri, Malak Inshas, Mohammad

Instructor: Dr.Akila sariee

Spring 2023

TABLE OF CONTENTS

Abstract:	2
Introduction:	2
Methodology:	3
part A:	3
Part b:	4
Part c:	5
Part d:	6
Part e:	8
Part f:	10
Part g:	11
Part h:	13
Flow charts:	15
For is_alpha:	15
is_digital:	15
to_lower:	16
to_upper:	16
find_str	17
replace_chr:	18
replace str:	18
Project timeline:	19
who did what:	19
Conclusion:	20

ABSTRACT:

This report provides an overview of programming functions and their applications. It describes the various categories of functions, such as built-in functions and user-defined functions, as well as how they can be used to accomplish particular duties. The report examines eight distinct functions, such as those used to manipulate strings and lists. In addition, the report provides examples of how to construct functions that manipulate strings and lists, such as functions that search for substrings, replace characters, and replace substrings within a string. It describes how these functions accept input parameters, transform input data, and return output.

INTRODUCTION:

In this project, we will develop an algorithm in Python to manage a library module of Python functions that we will have to write in order to manipulate strings and lists. Data classification and storage is an essential component of any program, but it is especially important when working with big amounts of information. Fortunately, we now have the assistance of computers and other forms of technology to assist us in working with such information, such as processing and organizing vast amounts of data. This solution will be written in need to manage the library module. A library is a collection of non-volatile resources that are utilized by computer programs, most commonly for the development of software. This concept comes from the field of computer science. These may consist of configuration data, documentation, help data, message templates, pre-written code and subroutines, classes, value specifications, or type specifications.

METHODOLOGY:

PART A:

```
def is_alpha(s):  
    for c in s:  
        if not c.isalpha():  
            return False  
    return True
```

1. This code defines the 'is_alpha()' function, which accepts a string's' as input and returns a Boolean value indicating whether the string contains only alphabetic characters.
2. The 'for' loop iterates over each 'c' character in the string's'. For each character 'c', the 'if' statement uses the 'isalpha()' method to determine whether 'c' is alphabetical. If 'c' is not alphabetical, the function returns 'False' to indicate that the string is not entirely alphabetical.
3. If all characters in the string are alphabetical, the function does not return 'False' within the 'for' loop and the 'return True' statement is executed outside of the loop, indicating that the string only contains alphabetical characters.
4. This code evaluates whether a given string contains only alphabetical characters and returns True if the string is wholly alphabetical and False otherwise.

The purpose of the code:

- The 'is_alpha()' function determines whether a given string contains only alphabetic characters, specifically upper- or lowercase ASCII letters.
- The function returns 'True' if all the characters in the input string are alphabetical, and 'False' otherwise

Input and output:

- Input = a string
- Output = returns as a boolean value(true or false)
 - true if the string contains only letters
 - False if there are any digits.

PART B:

```
def is_digit(d):  
    for c in d:  
        if not c.isdigit():  
            return False  
    return True
```

1. This code defines the 'is_digit()' function, which accepts the string 'd' as input and returns a Boolean indicating whether the string contains only digits.
2. The 'for' loop analyzes each occurrence of the character 'c' in the input string 'd'. The 'if' statement uses the 'isdigit()' method to determine whether each character 'c' is not a digit for each occurrence. If 'c' is not a digit, the function promptly returns 'False' to indicate that the string does not contain only digits.
3. If all characters in the string are digits, the function does not return 'False' during the 'for' iteration and the 'return True' statement is executed outside the loop to indicate that the string only contains digits.
4. This code checks whether a given string contains only digits and returns true if it does. True if the complete string consists of digits; False otherwise. This function may be useful when validating input sequences that are expected to contain only numeric values, such as age fields, phone number fields, and credit card number fields where non-numerical characters are prohibited. By utilizing this function, you can ensure that the input contains only digits, thereby reducing the likelihood of errors or security flaws resulting from unexpected input.

The purpose of the code:

- The 'is_digit()' function's job is to figure out whether or not a given string simply consists of numerical digits, more precisely ASCII decimal digits, and report its findings accordingly.

- The function determines whether or not all of the characters in the input string are digits and then either returns 'True' or 'False' depending on whether or not all of the characters are digits.

Input and output:

- Input = a string
- Out = returns a Boolean value
 - true if all of the strings include just numbers
 - false if a string contains both numbers and letters

PART C:

```
def to_lower(l):
    if l.lower():
        return l.lower()
    else:
        return l
```

1. This code defines a function named 'to_lower()' that accepts the string 'l' as input and returns a new string containing the input string in lowercase.
2. The function applies the 'lower()' method to the input string 'l' using dot notation, which converts all uppercase characters to lowercase.
3. The 'if' line then checks whether the lowercase version of the string is empty. If the lowercase version of the string is not empty, the function uses the 'return' line to return the lowercase version of the string.
4. If the lowercase version of the string is null (i.e., the input string was already lowercase), the function executes the 'else' statement and returns the original input string 'l'.

In general, this code converts a provided string to lowercase and verifies that it is not empty. The function returns the lowercase variant of the string if it is not empty, and the original input string if it is empty. This function can be utilized to perform case-insensitive string comparisons or to normalize the case of input strings during data preprocessing tasks.

The purpose of the code:

- The purpose of the 'to_lower()' function is to provide a straightforward and efficient method for converting a string to lowercase by replacing all uppercase characters with their lowercase counterparts.
- The function 'to_lower()' converts a string to lowercase by replacing all uppercase ASCII characters with their lowercase counterparts.
- This function is beneficial when you need to standardize the case of input strings in order to perform case-insensitive string operations, such as searching, categorizing, and comparing strings.
- In addition, this function can be used to standardize the case of input sequences to ensure consistency and accuracy in subsequent analysis or modeling.

Input and output:

- Input = a string
- Out = the string but in lowercase

PART D:

```
def to_upper(u):
    if u.upper():
        return u.upper()
    else:
        return u
```

1. The following line of code provides a function known as "to_upper()," which accepts a string with the value "u" as its input and then returns a new string that is an uppercase version of the string with which it was provided.
2. Using the dot notation, the function first applies the 'upper()' method to the input string 'u', which converts all lowercase characters to uppercase.
3. The 'if' statement then determines if the string in uppercase is null. If the uppercase version of the string is not null, the 'return' statement returns the uppercase version of

the string. If the uppercase version of the string is empty (i.e., the input string was already capitalized), the 'else' statement is executed and the function returns the original input string 'u'.

4. Overall, this code converts a provided string to uppercase while determining if it is not null. If the uppercase version of the string is not empty, the function returns the uppercase version of the string; otherwise, it returns the original string input. This function can be utilized to perform case-insensitive string comparisons or to normalize the case of input strings during data preprocessing tasks.

The purpose of this code:

- The 'to_upper()' function converts a given string to uppercase by replacing all lowercase ASCII characters with their respective uppercase characters.
- This function comes in handy when you need to standardize the case of input strings in order to perform case-insensitive string operations, such as searching, categorizing, and comparing strings.
- For example, if you are searching for an expression in a list of strings, you might want to convert all the strings to uppercase to ensure that the search is not case-sensitive. Similarly, if you are comparing two strings, you may want to convert them both to uppercase to ensure that case differences do not influence the comparison.
- In addition, this function can be used to standardize the case of input sequences to ensure consistency and accuracy in subsequent analysis or modeling.
- The purpose of the 'to_upper()' function is to provide a simple and efficient method of converting a string to uppercase by replacing all lowercase characters with their uppercase counterparts.

Input and output:

- Input = a string
- Out = the string but in uppercase\

PART E:

```
def find_chr(string, char):  
    if len(char) != 1 or char not in string:  
        return -1  
    else:  
        return string.find(char)|
```

1. The 'find_chr' function in Python This method requires two arguments: a string 'string' and a character 'char'. This function returns the index of the first occurrence of the character 'char' in the string 'string', or -1 if the character cannot be discovered or if 'char' is not a single character.
2. The first line of the function verifies if the length of 'char' is greater than 1 or if 'char' does not appear in the string 'string'. If either of these conditions is met, 'char' is either not a single character or is absent from the string. In this instance, the function returns -1 to signify that the specified character could not be located.
3. If the conditions in the first sentence are not met, it indicates that 'char' is a singular character that is present in the string. The 'else' block is executed, and the string object's 'find()' method is used to determine the index of the first occurrence of 'char' in the string. The function then returns the index as its result. Note that the 'find()' method returns the index of the substring's first occurrence in the string, or -1 if the substring is not detected. In this instance, we are seeking for a single character, which is considered a substring of length 1.

4. This function can be utilized to determine the index of a particular character within a string. The function returns -1 if the specified character is not found or if it is not a single character.

The purpose of the code:

- The 'find_chr' function searches a string for a single character and returns the index of its first occurrence. The function accepts two string parameters: 'string' and 'char'. Both parameters are strings. The function returns the index of the first occurrence of the character 'char' within the string 'string', or -1 if the character cannot be located or if 'char' is not a single character.
- This function is beneficial in a variety of situations where a specific character must be located within a string. Using this function, you could, for example, search for a specific character within a user-entered string or extract a substring from a larger string based on the position of a specific character.
- The function provides a straightforward and convenient method to search for a character within a string, and it can help to simplify string manipulation and search-related code.

Input and output:

- Input: a string + a character
- Output: index number
 - If the character is not detected, the output will be -1

PART F:

```
def find_str(string, sub_string):  
    if sub_string not in string:  
        return -1  
    else:  
        return string.find(sub_string)
```

1. This Python function 'find_str' accepts two arguments: a string 'string' and a substring 'sub_string'. The function returns the index of the first instance of the substring 'sub_string' within the string 'string', or -1 if the substring is not detected.
2. The function's first line determines whether the substring "sub_string" is missing from the string "string." If this condition evaluates to true, the substring is not present in the string. In this instance, the function returns -1 to signify that the specified substring was not located.
3. The substring is present in the string if the condition on the first line is false. The 'else' block is executed, and the string object's 'find()' method is used to determine the index of the substring's first occurrence. The index is then returned as the function's result. the 'find()' method returns the index of the first occurrence of the specified substring within the string, or -1 if the substring cannot be located.
4. Overall, this function can be used to locate the index of a substring within a string. If the specified substring cannot be located, the function returns -1. This function is beneficial in a variety of situations where a specific substring must be located within a larger string.

The purpose of the code:

- The function 'find_str' searches for a substring within a larger string and returns the index of the substring's first occurrence. The function accepts two string parameters: 'string'

and 'sub_string'. The function returns the index of the first appearance of the substring 'sub_string' in the string 'string', or -1 if the substring is not detected.

- This function is beneficial in a variety of situations where a specific substring must be located within a larger string. For instance, you could use this function to search for a particular word or phrase within a document or to extract a substring from a broader string based on the position of a particular word or phrase.
- The function provides a basic and convenient method for searching for a substring within a string, and it can aid in the simplification of code involving string manipulation and searching. By returning the index of the substring's first occurrence, this function enables easy extraction and manipulation of the substring.

Input and output:

- Input: string + substring
- Output: first occurrence of the substring
-if substring not in string return -1

PART G:

```
def replace_char(strr, str1, str2):  
    str_new = strr.replace(str1, str2)  
    return str_new
```

1. This Python function 'replace_char' accepts three arguments: a string 'strr', a string 'str1', and a string 'str2'. The function returns a new string that is identical to 'strr', except that all instances of 'str1' have been replaced with 'str2'.

2. The function uses the `'replace()'` method of the string object to create a new string `'str_new'` that is identical to `'strr'` but with all occurrences of the string `'str1'` replaced with the string `'str2'`.
3. The `'replace()'` method returns a modified version of the original string as a new string. The original string remains unchanged. The function then returns the new string `'str_new'` as its result.
4. This function can be used to replace all instances of a particular string within another string with another string. This can be beneficial in a variety of situations where replacing specific substrings with new values is required to modify a string.

The purpose of the code:

- When you need to replace a specific character within a string with another character, the function is useful. The function can modify the string as required by replacing all occurrences of the character in the string.
- If neither `'char1'` nor `'char2'` is a single character, the function returns an empty string to signify that the replacement could not be performed. This verification ensures that the function is used properly and that the inputs are legitimate.
- In general, the `'replace_chr'` function offers a convenient and simple method for replacing a single character within a string with another character, and can aid in the simplification of string manipulation and replacement code.

PART 8:

```
def replace_str(string, old_str, new_str):  
    str_new = string.replace(old_str, new_str)  
    return str_new
```

1. This Python function 'replace_str' accepts three arguments: a string 'string', a substring 'old_str', and a substring 'new_str'. The function returns a string identical to 'string' with all instances of the substring 'old_str' replaced by 'new_str'.
2. Using the 'replace()' method of the string object, the function creates a new string 'str_new' that is identical to 'string' but with all occurrences of the substring 'old_str' replaced with the substring 'new_str'.
3. The 'replace()' method returns a string that represents the modified version of the original string. There is no modification to the original string. The function then returns the new string 'str_new' as its result.
4. This function can be used to replace all instances of a substring within another string with a different substring. This can be useful in a variety of situations where it is necessary to modify a string by replacing specific substrings with new values. The function provides a straightforward and convenient method to perform this operation, which can help to simplify code involving string manipulation and replacement.

The purpose of this code:

- The 'replace_str' function is designed to replace all instances of a substring within another string with a new substring and return the modified string. The function returns a copy of the original string if the substring is not located in the string. If the substring to be

replaced is the empty string, the function also inserts the new substring before the first character, between each character, and after the last character of the original string.

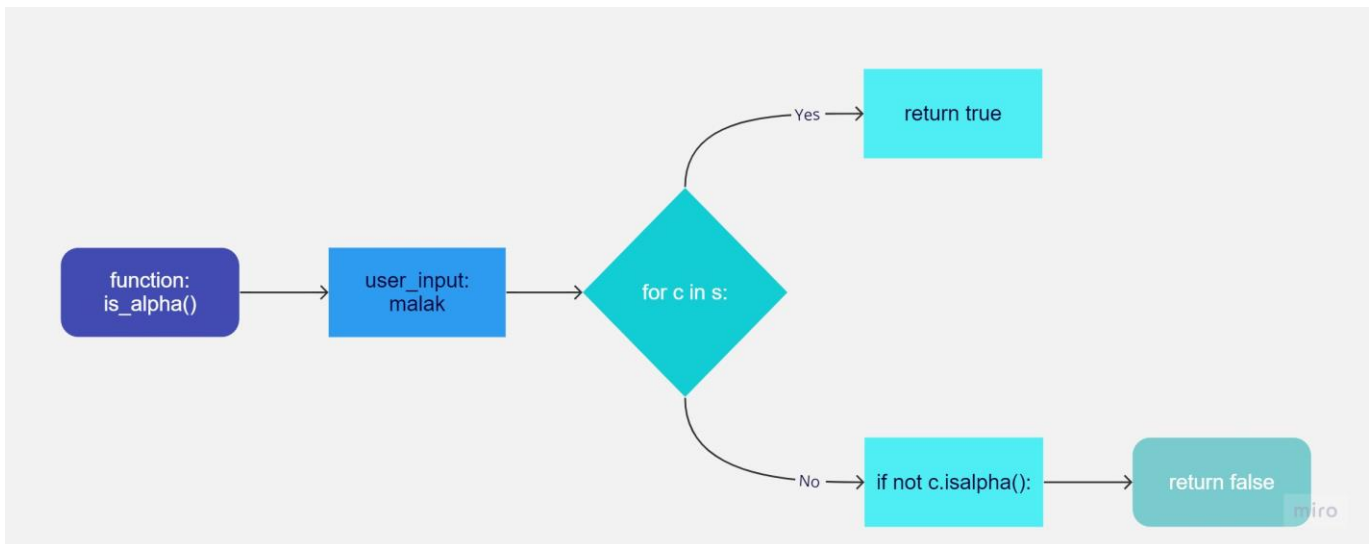
- For example Formatting data. When working with data, it is frequently necessary to format the data in accordance with the requirements of a specific system or application. For instance, you could use 'replace_str' to replace all instances of a particular character in a CSV file with another character, or to format a date string in a particular manner.
- When working with text, it is often necessary to modify the text in a variety of ways to satisfy the requirements of a specific application or use case. For instance, you could use 'replace_str' to modify the formatting of a block of text or to replace all instances of a specific word or phrase in a document with another word or phrase.
- Overall, the 'replace_str' function is a flexible utility that can be utilized in a vast array of applications where string manipulation and replacement is required. It offers a straightforward and convenient method for modifying strings and can make your code more efficient and simpler to comprehend.

Input and output:

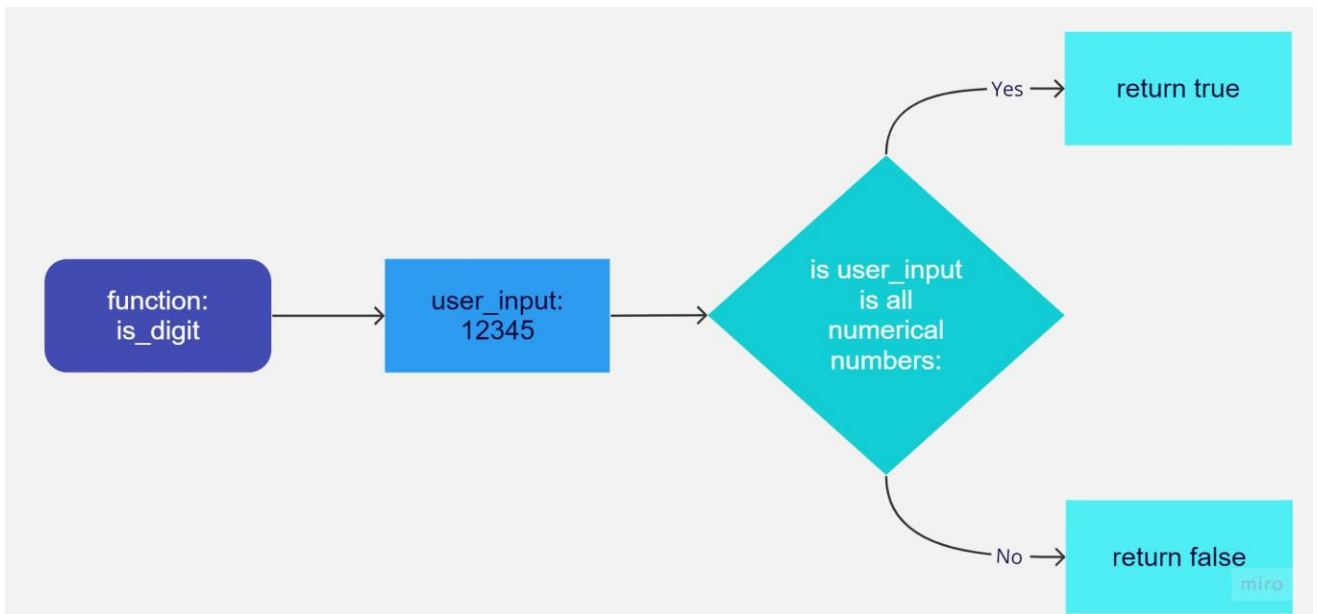
- Input: string + old substring + new substring
- Output: string containing new substring

FLOW CHARTS:

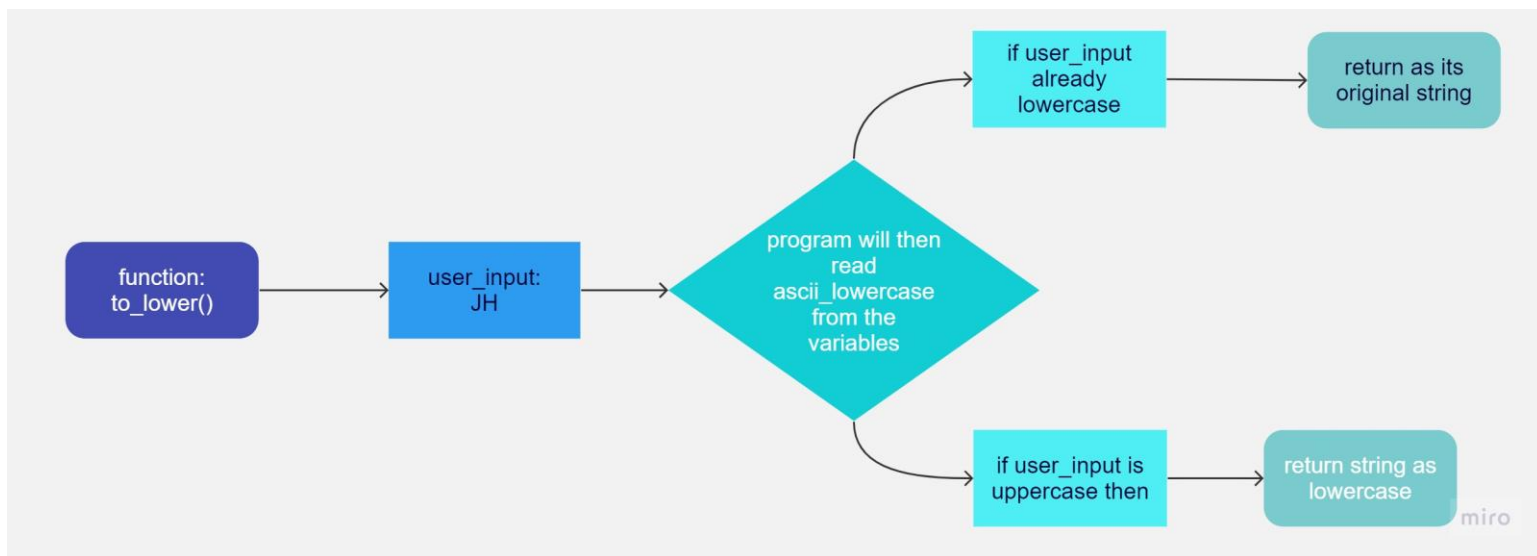
FOR IS_ALPHA:



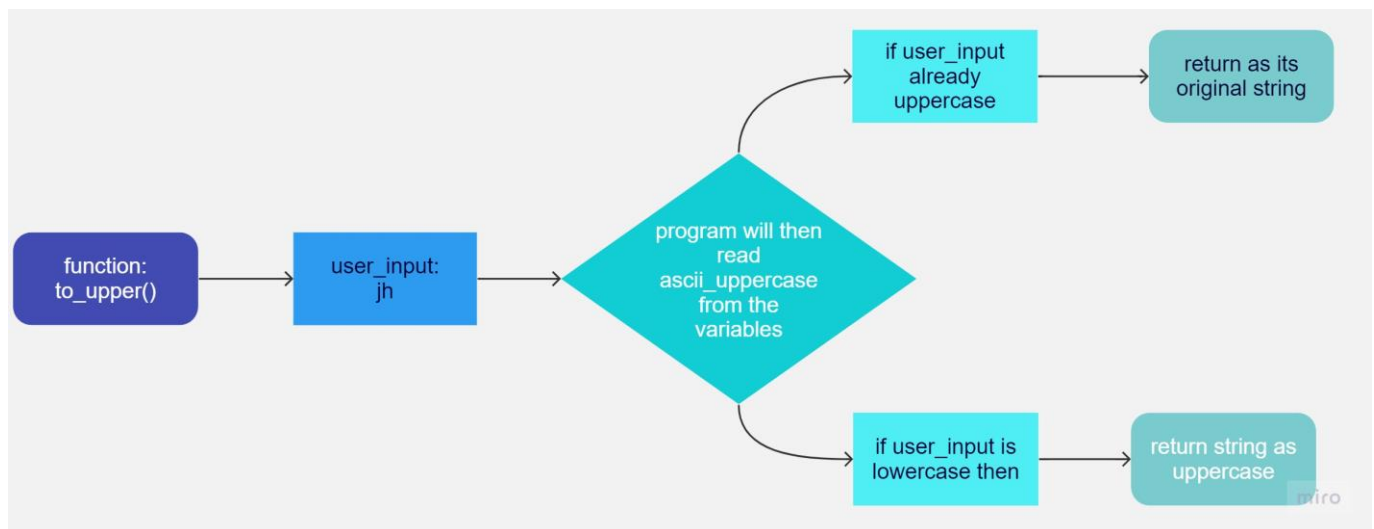
IS_DIGITAL:



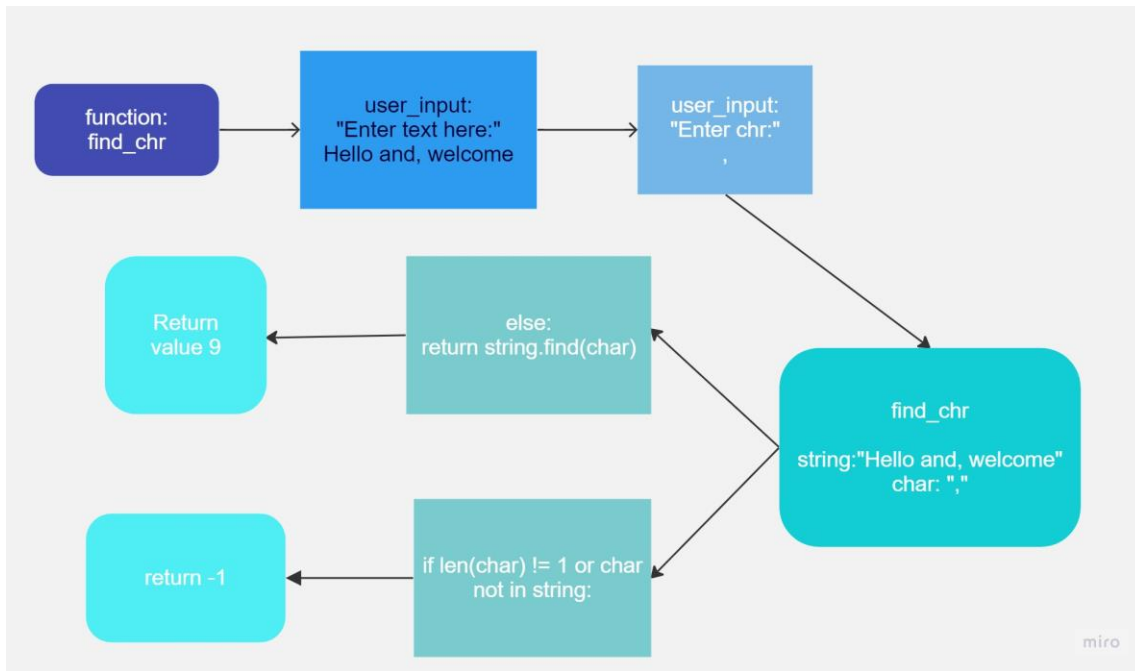
TO_LOWER:



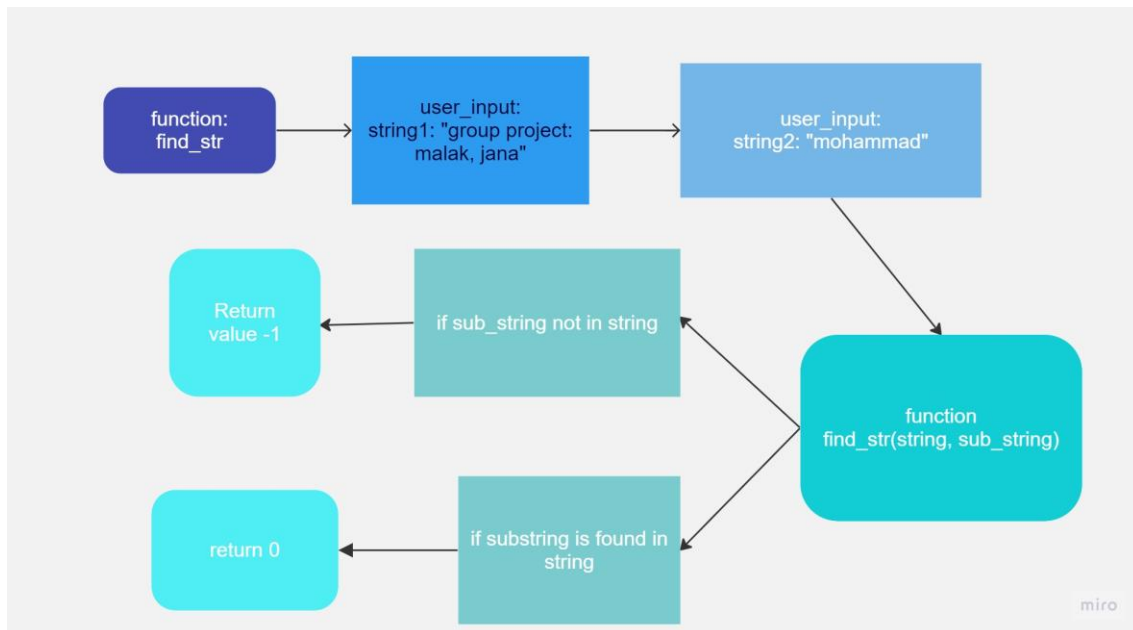
TO_UPPER:



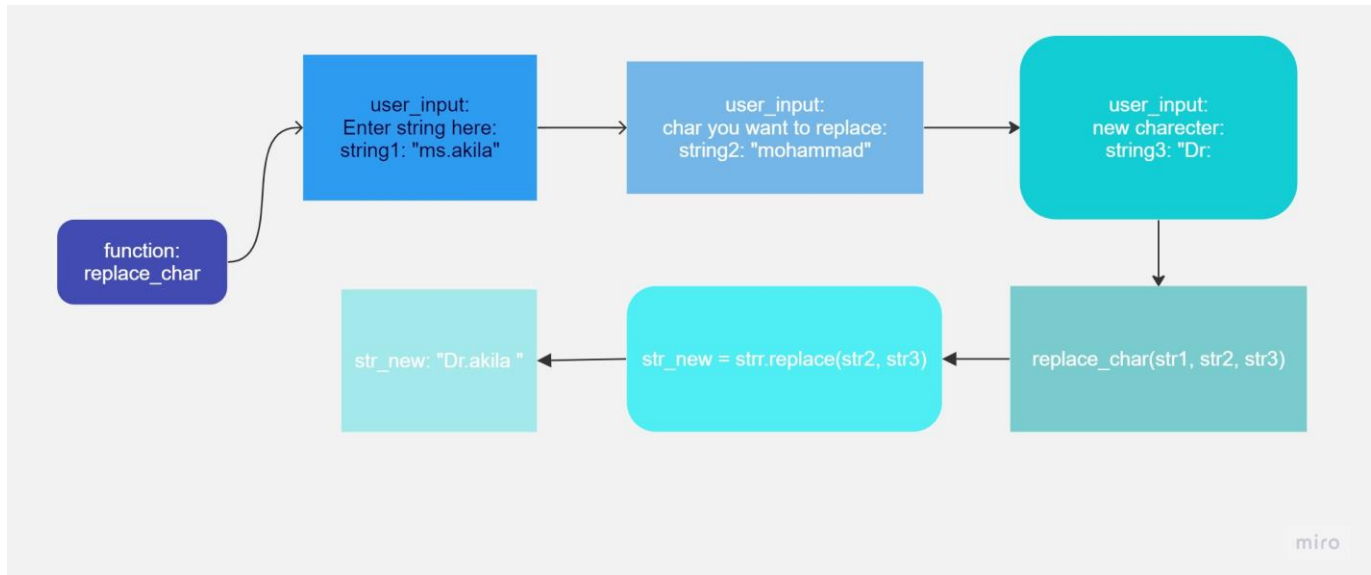
FIND_CHR:



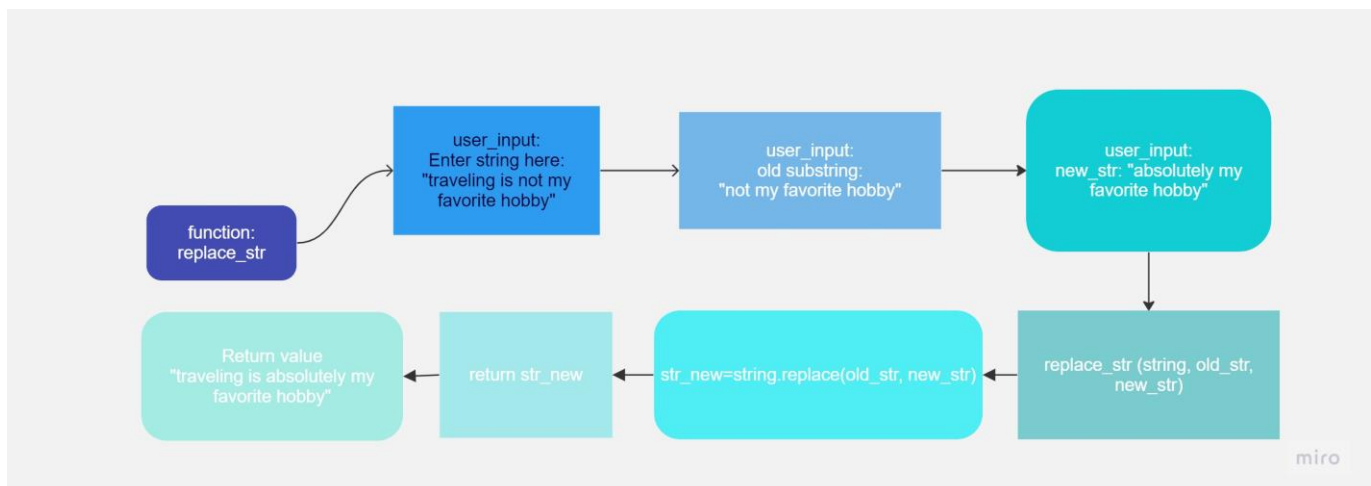
FIND_STR



REPLACE_CHR:



REPLACE_STR:



PROJECT TIMELINE:

8 may- start of our project and first disscustion

9 may-writing drafts of the codes

10 may till 14 may-working on the codes

15 may till 16 may- working on the testing

15 may till 18 may -report done and completed

19 may-powerpoint presentation completed

23 may- presentation day

WHO DID WHAT:

Parts of projects	Names:
Codes	Malak and Jana
testing	Jana
Methodology	Malak
report	Mohammad
PowerPoint presentation	Jana, Malak and, Mohammad

CONCLUSION:

Libraries and modules are potent programming tools that enable developers to reuse code and build upon existing functionality. By dividing a program into smaller functions and modules, we can generate code that is more manageable, efficient, and simple to maintain and modify.

By utilizing libraries, modules, and functions, we can create more efficient, maintainable, and adaptable programs overall. This method permits us to build upon existing code and generate more potent and complex programs with less effort and fewer errors by reusing existing code.