Q.454

Write a Python function to implement linear search algorithm. Linear search is also called as sequential search algorithm. It is the simplest searching algorithm. In Linear search, we simply traverse the list completely and match each element of the list with the item whose location is to be found. If the match is found, then the location of the item is returned; otherwise, the algorithm returns NULL. The following is linear search algorithm: Given a list L of n elements with values or records L0 .... Ln−1, and target value T, the following subroutine uses linear search to find the index of the target T in L.

1. Set i to 0.
2. If Li = T, the search terminates successfully; return i.
3. Increase i by 1.
4. If i < n, go to step 2. Otherwise, the search terminates unsuccessfully. Input: Enter the list of numbers: 5 4 3 2 1 10 11 2 The number to search for: 1 Output: 1 was found at index 4

In [1]:
```
def linear_search(alist, key):
    """Return index of key in alist. Return -1 if key not present."""
    for i in range(len(alist)):
        if alist[i] == key:
            return i
    return -1


alist = input('Enter the list of numbers: ')
alist = alist.split()
alist = [int(x) for x in alist]
key = int(input('The number to search for: '))

index = linear_search(alist, key)
if index < 0:
    print('{} was not found.'.format(key))
else:
    print('{} was found at index {}.'.format(key, index))
```

```
Enter the list of numbers: 5 4 3 2 1 10 11 2
The number to search for: 1
1 was found at index 4.
```

Q. 455.

Given a list of elements, write a python program to perform grouping of similar elements, as different key-value list in dictionary. Print the dictionary sorted in descending order of frequency of the elements. Note: To perform the sorting, use the sorted function by converting the dictionary into a list of tuples. After sorting, convert the list of tuples back into a dictionary and print it. Input : test_list = [4, 6, 6, 4, 2, 2, 4, 8, 5, 8] Output : {4: [4, 4, 4], 6: [6, 6], 2: [2, 2], 8: [8, 8], 5: [5]} Explanation : Similar items grouped together on occurrences. Input : test_list = [7, 7, 7, 7] Output : {7 : [7, 7, 7, 7]} Explanation : Similar items grouped together on occurrences

In [2]:
```
def group_elements(input_list):
    frequency_dict = {}

    # Group similar elements in the dictionary
    for element in input_list:
        if element not in frequency_dict:
            frequency_dict[element] = []
        frequency_dict[element].append(element)

    # Sort the dictionary by frequency in descending order
    sorted_tuples = sorted(frequency_dict.items(), key=lambda x: len(x[1]), reverse

    # Convert the sorted list of tuples back into a dictionary
    sorted_dict = dict(sorted_tuples)

    return sorted_dict

# Example usage
test_list = [4, 6, 6, 4, 2, 2, 4, 8, 5, 8]
result_dict = group_elements(test_list)

# Print the sorted dictionary
print(result_dict)
```

```
{4: [4, 4, 4], 6: [6, 6], 2: [2, 2], 8: [8, 8], 5: [5]}
```

Q.456.

A digital image in a computer is represented by a pixels matrix. Each image processing operation in a computer may be observed as an operation on the image matrix. Suppose you are given an N x N 2D matrix A (in the form of a list) representing an image. Write a Python program to rotate this image by 90 degrees (clockwise) by rotating the matrix 90 degree clockwise. Write proper code to take input of N from the user and then to take input of an N x N matrix from the user. Rotate the matrix by 90 degree clockwise and then print the rotated matrix. Note: You are not allowed to use an extra iterable like list, tuple, etc. to do this. You need to make changes in the given list A itself. Your program should be able to handle any N x N matrix from N = 1 to N = 20

In [8]:
```
def rotate_matrix(matrix, n):
    # Transpose the matrix
    for i in range(n):
        for j in range(i + 1, n):
            matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]

    # Reverse each row
    for i in range(n):
        matrix[i].reverse()

# Get input from the user for N
n = int(input("Enter the size of the matrix (N): "))

# Get input from the user for the matrix
matrix = []
print("Enter the matrix (each row on a new line):")
for _ in range(n):
    row = list(map(int, input().split()))
    matrix.append(row)
```

```
# Rotate the matrix by 90 degrees clockwise
rotate_matrix(matrix, n)

# Print the rotated matrix
print("\nRotated Matrix:")
for row in matrix:
    print(*row)
```

```
Enter the size of the matrix (N): 3
Enter the matrix (each row on a new line):
1 2 3
4 5 6
7 8 9

Rotated Matrix:
7 4 1
8 5 2
9 6 3
```

Q. 457.

Write a Program to Print Longest Common Prefix from a given list of strings. The longest common prefix for list of strings is the common prefix (starting of string) between all strings. For example, in the given list [“apple”, “ape”, “zebra”], there is no common prefix because the 2 most dissimilar strings of the list “ape” and “zebra” do not share any starting characters. If there is no common prefix between all strings in the list than return -1. For example, Input list: ["lessonplan", "lesson","lees", "length"] The longest Common Prefix is: le Input list: ["python","pythonprogramming","pythonlist"] The longest Common Prefix is: python Input list: ["lessonplan", "lesson","ees", "length"] The longest Common Prefix is: -1

In [9]:
```
def longest_common_prefix(str_list):
    if not str_list:
        return -1

    # Sort the list to get the most dissimilar strings at the ends
    str_list.sort()

    # Check the common prefix between the first and last strings
    prefix = ""
    for i in range(len(str_list[0])):
        if str_list[0][i] == str_list[-1][i]:
            prefix += str_list[0][i]
        else:
            break

    # If no common prefix found, return -1
    return prefix if prefix else -1

# Example usage
input_list1 = ["lessonplan", "lesson", "lees", "length"]
result1 = longest_common_prefix(input_list1)
print("The longest Common Prefix is:", result1)

input_list2 = ["python", "pythonprogramming", "pythonlist"]
result2 = longest_common_prefix(input_list2)
print("The longest Common Prefix is:", result2)

input_list3 = ["lessonplan", "lesson", "ees", "length"]
```

```
result3 = longest_common_prefix(input_list3)
print("The longest Common Prefix is:", result3)
```

```
The longest Common Prefix is: le
The longest Common Prefix is: python
The longest Common Prefix is: -1
```

Q. 458

In [5]:

```
import math

def main():
    message = input("Enter message: ")
    key = int(input("Enter key [2-%s]: " % (len(message) - 1)))
    mode = input("Encryption/Decryption [e/d]: ")

    if mode.lower().startswith("e"):
        text = encryptMessage(key, message)
    elif mode.lower().startswith("d"):
        text = decryptMessage(key, message)

    # Append pipe symbol (vertical bar) to identify spaces at the end.
    print("Output:\n%s" % (text + "|"))


def encryptMessage(key, message):
    cipherText = [""] * key
    for col in range(key):
        pointer = col
        while pointer < len(message):
            cipherText[col] += message[pointer]
            pointer += key
    return "".join(cipherText)


def decryptMessage(key, message):
    numCols = math.ceil(len(message) / key)
    numRows = key
    numShadedBoxes = (numCols * numRows) - len(message)
    plainText = [""] * numCols
    col = 0
    row = 0

    for symbol in message:
        plainText[col] += symbol
        col += 1

        if (
            (col == numCols)
            or (col == numCols - 1)
            and (row >= numRows - numShadedBoxes)
        ):
            col = 0
            row += 1

    return "".join(plainText)


if __name__ == "__main__":
    import doctest
```

```
    doctest.testmod()
    main()
```

```
Enter message: Hloe gl o  sogrilw g epntstfii o yotay hee nnh aoiortiimreegehrun n
hnse ne
Enter key [2-73]: 5
Encryption/Decryption [e/d]: d
Output:
Hi hello how are you going to learn python in this semester of engineering|
```

Q. 459

Write a python program to print all possible combinations from the three Digits and also count unique values inside a list and also find list product excluding duplicates and also find sum of list's elements excluding duplicates. Examples: To print all possible combinations Input: [1, 2, 3] Output: 1 2 3 1 3 2 2 1 3 2 3 1 3 1 2 3 2 1 Count unique values inside a list input = [1, 2, 3] No of unique items are: 3 input = [1, 2, 2] No of unique items are: 2 input = [2, 2, 2] No of unique items are: 3

List product excluding duplicates Input: [2, 3, 5] Duplication removal list product: 30 Input: [2, 2, 3] Duplication removal list product: 6 Sum of list's elements excluding duplicates Input: [1, 3, 5] Output: 9 Input: [1, 2, 2] Output: 3

In [2]:
```
def generate_permutations(digits, current_index, result):
    if current_index == len(digits) - 1:
        result.append(digits[:])  # Append a copy of the list
    else:
        for i in range(current_index, len(digits)):
            digits[current_index], digits[i] = digits[i], digits[current_index]
            generate_permutations(digits, current_index + 1, result)
            digits[current_index], digits[i] = digits[i], digits[current_index]

def print_combinations(digits):
    result = []
    generate_permutations(digits, 0, result)
    for perm in result:
        print(*perm)

def count_unique_items(lst):
    unique_items = set(lst)
    print(f"No of unique items are: {len(unique_items)}")

def product_excluding_duplicates(lst):
    unique_items = set(lst)
    product = 1
    for item in unique_items:
        product *= item
    print(f"Duplication removal list product: {product}")

def sum_excluding_duplicates(lst):
    unique_items = set(lst)
    total_sum = sum(unique_items)
    print(f"Sum of list's elements excluding duplicates: {total_sum}")

# Examples
digits_combinations = [1, 2, 3]
print("To print all possible combinations:")
print_combinations(digits_combinations)
```

```
print("\nCount unique values inside a list:")
count_unique_items([1, 2, 3])
count_unique_items([1, 2, 2])
count_unique_items([2, 2, 2])

print("\nList product excluding duplicates:")
product_excluding_duplicates([2, 3, 5])
product_excluding_duplicates([2, 2, 3])

print("\nSum of list's elements excluding duplicates:")
sum_excluding_duplicates([1, 3, 5])
sum_excluding_duplicates([1, 2, 2])
```

```
To print all possible combinations:
1 2 3
1 3 2
2 1 3
2 3 1
3 2 1
3 1 2

Count unique values inside a list:
No of unique items are: 3
No of unique items are: 2
No of unique items are: 1

List product excluding duplicates:
Duplication removal list product: 30
Duplication removal list product: 6

Sum of list's elements excluding duplicates:
Sum of list's elements excluding duplicates: 9
Sum of list's elements excluding duplicates: 3
```

Q. 460

• Use appropriate comment lines to divide subprograms. • Also demonstrate the program with one example test case. (Example test input and output are given)

PART - A Using map function, write a Python program to convert the given list into a tuple of strings. For the given input, the program must print the output as shown below - Input – [1,2,3,4] Output – ('1','2','3','4')

PART - B Write a Python program that multiply each number of the given list with 10 using lambda function. For the given input, the program must print the output as shown below - Input – [1,2,3,4] Output – [10,20,30,40]

PART - C Write a Python program that multiply all elements of the given list using reduce function and return the product. For the given input, the program must print the output as shown below - Input – [1,2,3,4] Output – 24 (which is 1*2*3*4)

PART - D Write a Python program satisfying following conditions - Create a python function countchar() that count the character of a string in a given string without using inbuilt functions. For the given input, the program must print the output as shown below -

Given input string – 'hello' countchar('l') Output : 2

 Create a python function findchar() that find the index of first occurrence of a character in a given string without using inbuilt functions. It should return -1 if it does not find the character. For the given input, the program must print the output as shown below -

Given input string – 'helloe' findchar('e') Output : 1 findchar('z') Output : -1

In [3]:
```
from functools import reduce

# PART - A
def convert_to_tuple(lst):
    return tuple(map(str, lst))

# PART - B
multiply_by_10 = lambda x: x * 10

# PART - C
def multiply_all_elements(lst):
    return reduce(lambda x, y: x * y, lst)

# PART - D
def countchar(s, char):
    count = 0
    for c in s:
        if c == char:
            count += 1
    return count

def findchar(s, char):
    for i, c in enumerate(s):
        if c == char:
            return i
    return -1

# Example test case
# PART - A
input_list = [1, 2, 3, 4]
output_tuple = convert_to_tuple(input_list)
print(f"PART - A\nInput: {input_list}\nOutput: {output_tuple}\n")

# PART - B
output_multiply_by_10 = list(map(multiply_by_10, input_list))
print(f"PART - B\nInput: {input_list}\nOutput: {output_multiply_by_10}\n")

# PART - C
output_multiply_all_elements = multiply_all_elements(input_list)
print(f"PART - C\nInput: {input_list}\nOutput: {output_multiply_all_elements}\n")

# PART - D
input_string = 'helloe'
char_to_count = 'l'
char_to_find_1 = 'e'
char_to_find_2 = 'z'

count_result = countchar(input_string, char_to_count)
print(f"PART - D\nGiven input string: '{input_string}'\ncountchar('{char_to_count}'

find_result_1 = findchar(input_string, char_to_find_1)
print(f"findchar('{char_to_find_1}'): Output: {find_result_1}")

find_result_2 = findchar(input_string, char_to_find_2)
print(f"findchar('{char_to_find_2}'): Output: {find_result_2}")
```

```
PART - A
Input: [1, 2, 3, 4]
Output: ('1', '2', '3', '4')

PART - B
Input: [1, 2, 3, 4]
Output: [10, 20, 30, 40]

PART - C
Input: [1, 2, 3, 4]
Output: 24

PART - D
Given input string: 'helloe'
countchar('l'): Output: 2

findchar('e'): Output: 1
findchar('z'): Output: -1
```

Q. 461

Write a Python program to calculate the sum of the positive and negative numbers of the below given list of numbers using lambda function. Input : m = [2, 4, -6, -9, 11, -12, 14, -5, 17] Output : Sum of the positive numbers: -32 Sum of the negative numbers: 48

In [5]:
```
m = [2, 4, -6, -9, 11, -12, 14, -5, 17]

positive_sum = sum(filter(lambda x: x > 0, m))
negative_sum = sum(filter(lambda x: x < 0, m))

print(f"Sum of the positive numbers: {positive_sum}")
print(f"Sum of the negative numbers: {negative_sum}")
```

```
Sum of the positive numbers: 48
Sum of the negative numbers: -32
```

Q. 462

Create a python program which takes password as input and a function which checks whether the given password is valid or not under following conditions without using the RegEx module in Python language. Conditions required for a valid password:

1. Password strength should be at least 8 characters long
2. Password should contain at least one uppercase and one lowercase character.
3. Password must have at least one number

In [10]:
```
def is_valid_password(password):
    # Condition 1: Password should be at least 8 characters long
    if len(password) < 8:
        return False

    # Condition 2: Password should contain at least one uppercase and one lowercase
    has_uppercase = False
    has_lowercase = False

    for char in password:
```

```
        if char.isupper():
            has_uppercase = True
        elif char.islower():
            has_lowercase = True

    if not (has_uppercase and has_lowercase):
        return False

    # Condition 3: Password must have at least one number
    has_digit = False

    for char in password:
        if char.isdigit():
            has_digit = True

    return has_digit

# Example usage
user_password = input("Enter your password: ")

if is_valid_password(user_password):
    print("Password is valid!")
else:
    print("Password is invalid. Please follow the specified conditions.")
```

```
Enter your password: arch@123
Password is invalid. Please follow the specified conditions.
```

Q. 463

Write a python program with user defined function that reads the words from paragraph and stores them as keys in a dictionary and count the frequency of it as a value . For Example: Input string: “Dog the quick brown fox jumps over the lazy dog” Output: {'the': 2, 'jumps': 1, 'brown': 1, 'lazy': 1, 'fox': 1, 'over': 1, 'quick': 1, 'dog': 2}

In [11]:
```
def count_word_frequency(paragraph):
    words = paragraph.split()
    word_frequency = {}

    for word in words:
        # Convert to lowercase to ensure case-insensitive counting
        word = word.lower()

        # Remove punctuation marks (if any)
        word = word.strip(".,!?")

        if word in word_frequency:
            word_frequency[word] += 1
        else:
            word_frequency[word] = 1

    return word_frequency

# Example usage
paragraph_input = "Dog the quick brown fox jumps over the lazy dog"
result_dict = count_word_frequency(paragraph_input)

print("Input string:", paragraph_input)
print("Output:", result_dict)
```

```
Input string: Dog the quick brown fox jumps over the lazy dog
Output: {'dog': 2, 'the': 2, 'quick': 1, 'brown': 1, 'fox': 1, 'jumps': 1, 'over':
1, 'lazy': 1}
```

Q. 464

Write a python Program to check entered password by user is correct or not. Entered password is correct if it has upper character, lower character , digits (but not more than 3 digits) ,special character and length is greater than or equal to eight and less than equal to fifteen. Get the digits from entered password and convert it in to number and then convert it in to English Word . Example: case 1 pw= R@m@3fa1tu9e $ValidPasswordnum = 319threehundredandnineteencase2pw = S@m@6a1tue$ Valid Password num= 61 sixty-one case 3 pw= S@m@6a26u8$ Invalid Password

In [18]:
```python
def is_valid_password(password):
    upper_found = False
    lower_found = False
    digit_count = 0
    special_found = False

    for char in password:
        if char.isupper():
            upper_found = True
        elif char.islower():
            lower_found = True
        elif char.isdigit():
            digit_count += 1
        elif char in "!@#$%^&*()_+-=[]{}|;:'\",.<>/?`~":
            special_found = True

    return (
        upper_found and
        lower_found and
        1 <= digit_count <= 3 and
        special_found and
        8 <= len(password) <= 15
    )

def convert_to_words(num):
    ones = ["", "one", "two", "three", "four", "five", "six", "seven", "eight", "ni
    teens = ["", "eleven", "twelve", "thirteen", "fourteen", "fifteen", "sixteen",
    tens = ["", "ten", "twenty", "thirty", "forty", "fifty", "sixty", "seventy", "e

    if 1 <= num < 10:
        return ones[num]
    elif 11 <= num < 20:
        return teens[num - 10]
    elif 10 <= num < 100:
        return tens[num // 10] + ("-" + ones[num % 10] if num % 10 != 0 else "")
    elif num == 100:
        return "one hundred"
    elif 100 < num < 1000:
        return ones[num // 100] + " hundred and " + convert_to_words(num % 100)
    else:
        return "Invalid number"

# Example usage
password_input = input("Enter your password: ")
```

```
if is_valid_password(password_input):
    print("Valid Password")

    # Extract digits from the password
    digits = ""
    for char in password_input:
        if char.isdigit():
            digits += char

    # Convert digits to a number and then to English words
    if digits:
        num = int(digits)
        print(f"num= {num}")
        words = convert_to_words(num)
        print(words)
    else:
        print("No valid number in the password")
else:
    print("Invalid Password")
```

```
Enter your password: S@m@6a26u8$
Invalid Password
```

Q. 465

Write a Python Program using function to count number of strings where the string length is 3 or more and the first and last character are same from a given list of string. Example : Input: ['abc','xyz','aba','2112','123451','12345'] Output: 3

In [19]:
```
def count_strings_with_same_first_last(lst):
    count = 0
    for string in lst:
        if len(string) >= 3 and string[0] == string[-1]:
            count += 1
    return count

# Example usage
input_strings = ['abc', 'xyz', 'aba', '2112', '123451', '12345']
result_count = count_strings_with_same_first_last(input_strings)

print("Input:", input_strings)
print("Output:", result_count)
```

```
Input: ['abc', 'xyz', 'aba', '2112', '123451', '12345']
Output: 3
```

In [ ]:
```
# Q. 466
Given a list L of size N. You need to count the number of special elements in the g
of that element makes the list balanced. The list will be balanced if sum of even i
elements. Also print the updated lists after removal of special elements.
Example 1:
Input:
L=[5, 5, 2, 5, 8]
Output:
Original List: [5, 5, 2, 5, 8]
Index to be removed is: 0
List after removing index 0 : [5, 2, 5, 8]
Original List: [5, 5, 2, 5, 8]
Index to be removed is: 1
List after removing index 1 : [5, 2, 5, 8]
Total number of special elements: 2
```

```
Explaination:
If we delete L[0] or L[1], list will be balanced.
[5, 2, 5, 8]
(5+5) = (2+8)
So L[0] and L[1] are special elements, So Count is 2.
After removal of the special elements, list will be: [5, 2, 5, 8]
Example 2:
Input:
L=[2,1,6,4]
Output:
Original List: [2, 1, 6, 4]
Index to be removed is: 1
List after removing index 1 : [2, 6, 4]
Total Number of Special elements: 1
Explaination:
If we delete L[1] from list : [2,6,4]
(2+4) = (6)
Here only 1 special element. So Count is 1.
After removal of special element, list will be : [2,6,4]
```

In [20]:

```python
def is_balanced(lst):
    sum_even = sum(lst[::2])
    sum_odd = sum(lst[1::2])
    return sum_even == sum_odd

def count_and_remove_special_elements(L):
    special_count = 0

    for i in range(len(L)):
        temp_list = L[:i] + L[i+1:]
        if is_balanced(temp_list):
            special_count += 1
            print(f"Original List: {L}")
            print(f"Index to be removed is: {i}")
            print(f"List after removing index {i} : {temp_list}")

    return special_count

# Example usage
L1 = [5, 5, 2, 5, 8]
result_count1 = count_and_remove_special_elements(L1)
print(f"Total number of special elements: {result_count1}\n")

L2 = [2, 1, 6, 4]
result_count2 = count_and_remove_special_elements(L2)
print(f"Total number of special elements: {result_count2}")
```

```
Original List: [5, 5, 2, 5, 8]
Index to be removed is: 0
List after removing index 0 : [5, 2, 5, 8]
Original List: [5, 5, 2, 5, 8]
Index to be removed is: 1
List after removing index 1 : [5, 2, 5, 8]
Total number of special elements: 2

Original List: [2, 1, 6, 4]
Index to be removed is: 1
List after removing index 1 : [2, 6, 4]
Total number of special elements: 1
```

Q. 467

Write Python Program to create a dictionary with the key as the first character and value as a list of words starting with that character. Example: Input: Don't wait for your feelings to change to take the action. Take the action and your feelings will change Output: {'D': ['Don't'], 'w': ['wait', 'will'], 'f': ['for', 'feelings', 'feelings'], 'y': ['your', 'your'], 't': ['to', 'to', 'take', 'the', 'the'], 'c': ['change', 'change'], 'a': ['action.', 'action', 'and'], 'T': ['Take']}

In [21]:
```
def create_char_dict(sentence):
    words = sentence.split()
    char_dict = {}

    for word in words:
        first_char = word[0]
        if first_char.isalpha():
            if first_char not in char_dict:
                char_dict[first_char] = [word]
            else:
                char_dict[first_char].append(word)

    return char_dict

# Example usage
input_sentence = "Don't wait for your feelings to change to take the action. Take t
output_dict = create_char_dict(input_sentence)

print("Input:", input_sentence)
print("Output:", output_dict)
```

```
Input: Don't wait for your feelings to change to take the action. Take the action
and your feelings will change
Output: {'D': ['Don't'], 'w': ['wait', 'will'], 'f': ['for', 'feelings', 'feeling
s'], 'y': ['your', 'your'], 't': ['to', 'to', 'take', 'the', 'the'], 'c': ['chang
e', 'change'], 'a': ['action.', 'action', 'and'], 'T': ['Take']}
```

Q. 468

d={"student0":'Student@0',"student1":'Student@11',"student2":'Student@121', "student3":'Student@052',"student4":'Student@01278',"student5":'Student@0125', Student6":'Student@042', "student7":'Student@07800',"student8":'Student@012', "student9":'Student@04789'} Write a python program to update the password of any user given the above dictionary(d) which stores the username as the key of the dictionary and the username's password as the value of the dictionary. print the updated dictionary and print the username and password according to ascending order of password length of the updated dictionary. For the password updating of that username follow some instructions. and password then display "enter correct password and username". if the user does not enter the correct username and password in a given three chances then display "enter correct password and username" and "try after 24h" password to update the password of that username. If the user enters a new password not follow the below format, then display "follow the password format". if the user does not enter the password in a given format in a given three chances, then display "follow the password format" and "try after 24h" The check, of whether the new password format is correct or wrong makes the user define a function. That user define a function to return True or False for password valid or not. That user define function return value used in this program for new password validation. o New password must have the below format:

1. at least 1 number between 0 and 9
    A. at least 1 upper letter (between a and z)
    B. at least 1 lower letter (between A and Z)
    C. at least 1 special character out of @$_
    D. minimum length of the password is 8 and the maximum length is 15
    E. Do not use space and other special characters. Only uses @$_ If the new password follows the format of the password in a given three chances. then print the updated dictionary and print the username and password according to ascending order of password length of an updated dictionary. If the dictionary is not updated then take the old dictionary

In [23]:
```
def is_valid_password(new_password):
    has_digit = False
    has_upper = False
    has_lower = False
    has_special = False

    for char in new_password:
        if char.isdigit():
            has_digit = True
        if char.isupper():
            has_upper = True
        if char.islower():
            has_lower = True
        if char in '@$':
            has_special = True

    return (
        has_digit and
        has_upper and
        has_lower and
        has_special and
        8 <= len(new_password) <= 15 and
        password_contains_valid_chars(new_password)
    )

def password_contains_valid_chars(password):
    valid_chars = set("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01234567
    return all(char in valid_chars for char in password)

def update_password(username, current_password):
    attempts = 3

    while attempts > 0:
        new_password = input(f"Enter new password for {username}: ")

        if is_valid_password(new_password):
            d[username] = new_password
            print("Password updated successfully!")
            return

        print("Follow the password format. Try again.")
        attempts -= 1

    print("Try after 24h. Password not updated.")

# Given dictionary
d = {
    "student0": 'Student@0',
    "student1": 'Student@11',
```

```
    "student2": 'Student@121',
    "student3": 'Student@052',
    "student4": 'Student@01278',
    "student5": 'Student@0125',
    "student6": 'Student@042',
    "student7": 'Student@07800',
    "student8": 'Student@012',
    "student9": 'Student@04789'
}

# Updating password for a specific user (for example, "student1")
update_password("student1", d["student1"])

# Displaying the updated dictionary
print("Updated Dictionary:", d)

# Displaying usernames and passwords in ascending order of password length
sorted_credentials = sorted(d.items(), key=lambda x: len(x[1]))
print("\nUsernames and Passwords in Ascending Order of Password Length:")
for username, password in sorted_credentials:
    print(f"{username}: {password}")
```

```
Enter new password for student1: a
Follow the password format. Try again.
Enter new password for student1: b
Follow the password format. Try again.
Enter new password for student1: b
Follow the password format. Try again.
Try after 24h. Password not updated.
Updated Dictionary: {'student0': 'Student@0', 'student1': 'Student@11', 'student
2': 'Student@121', 'student3': 'Student@052', 'student4': 'Student@01278', 'studen
t5': 'Student@0125', 'student6': 'Student@042', 'student7': 'Student@07800', 'stud
ent8': 'Student@012', 'student9': 'Student@04789'}

Usernames and Passwords in Ascending Order of Password Length:
student0: Student@0
student1: Student@11
student2: Student@121
student3: Student@052
student6: Student@042
student8: Student@012
student5: Student@0125
student4: Student@01278
student7: Student@07800
student9: Student@04789
```

In [ ]:

```
def encrypt_message(message, key):
    encrypted_message = ''

    for i in range(key):
        group = message[i::key]
        encrypted_message += ''.join(group) + ' '

    return encrypted_message.strip()

def decrypt_message(encrypted_message, key):
    decrypted_message = ''

    for i in range(key):
        decrypted_message += encrypted_message[i::key]

    return decrypted_message

def create_char_dict(decrypted_message):
```