Q1

CODE

```
import re
def isValidPan(stri):
    if (re.fullmatch("^[A-Z]{4}[0-9]{1}[A-Z]{1}",stri) is not None):
        return True
    else:
        return False
def main():
    usr = input("Enter your Pan card: ")
    if isValidPan(usr):
        print("valid")
    else:
        print("not valid")
if __name__=="__main__":
    main()
```

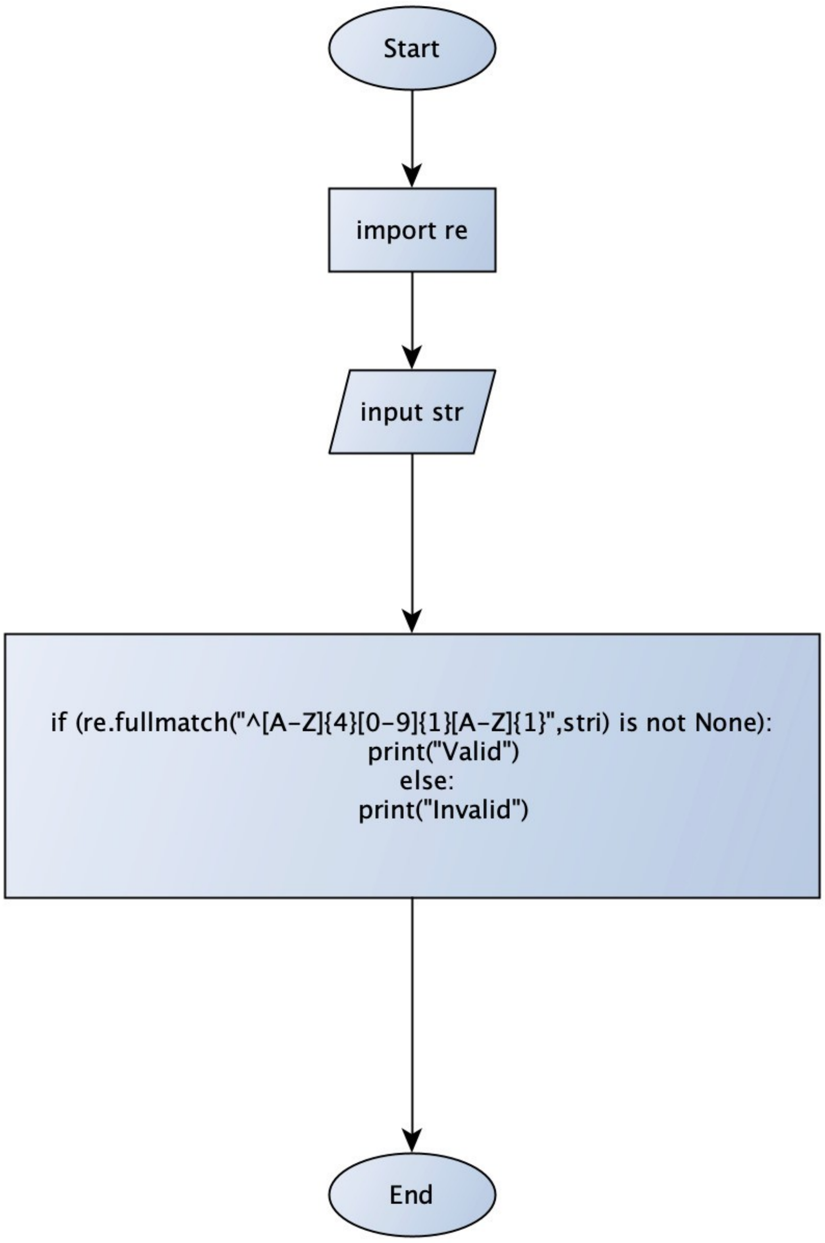
ALGO

- **Define Validation Function**:

- Define a function `isValidPan(stri)` that takes a string `stri` as input.
- Use a regular expression to check if the `stri` matches the required PAN format:
  - `^[A-Z]{4}[0-9]{1}[A-Z]{1}`
  - This pattern ensures:
    - The string starts with four uppercase letters `[A-Z]{4}`
    - Followed by a single digit `[0-9]{1}`
    - Ends with a single uppercase letter `[A-Z]{1}`
- If the string matches this pattern, return `True` (indicating it's valid).
- Otherwise, return `False`.

- **Main Program**:

- Define a `main()` function to handle user interaction.
- Prompt the user to enter their PAN card number using `input()`.
- Call `isValidPan()` with the input string.
- Print "valid" if `isValidPan()` returns `True`.
- Print "not valid" if `isValidPan()` returns `False`.


Start
import re
input str
if (re.fullmatch("^[A-Z]{4}[0-9]{1}[A-Z]{1}",stri) is not None):
print("Valid")
else:
print("Invalid")
End

```
import re
def isValidPass(stri):
    new_var = len(re.findall("[A-Z]",stri)) !=0
    new_var1 = len(re.findall("[a-z]",stri)) !=0
    new_var2 = len(re.findall("[0-9]",stri)) !=0
    new_var3 = len(re.findall("[_,@,$]",stri)) !=0
    print(new_var)
    if (new_var)  and (new_var1) and (new_var2) and (new_var3) and 8<=len(stri)<=16:
        return True
    else:
        return False
def main():
    usr = input("Enter your Password: ")
    if isValidPass(usr):
        print("valid")
    else:
        print("not valid")
if __name__=="__main__":
    main()
```

ALGO

- **Define Password Validation Function**:

- Define a function `isValidPass(stri)` that takes a string `stri` (the password) as input.
- Use `re.findall()` to check for the following conditions in the password:
  - `new_var`: Check if there is at least one uppercase letter (`[A-Z]`) in `stri`.
  - `new_var1`: Check if there is at least one lowercase letter (`[a-z]`) in `stri`.
  - `new_var2`: Check if there is at least one digit (`[0-9]`) in `stri`.
  - `new_var3`: Check if there is at least one special character (`[_,@,$]`) in `stri`.
- Each of these variables will be `True` if the corresponding condition is met and `False` otherwise.
- Print the value of `new_var` to help with debugging.

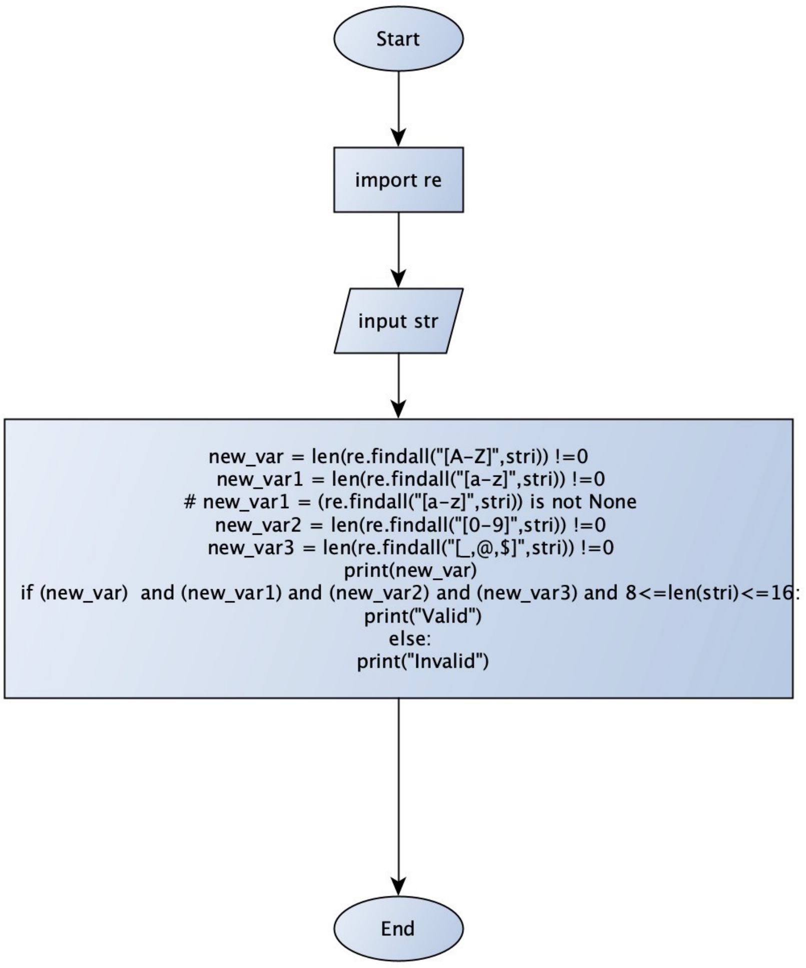
- **Combine Validation Conditions**:

- Check if all conditions are met:
  - `new_var`, `new_var1`, `new_var2`, and `new_var3` are all `True`
  - The length of `stri` is between 8 and 16 characters (inclusive).
- If all conditions are satisfied, return `True` (indicating the password is valid).
- Otherwise, return `False`.

- **Main Program**:

- Define a `main()` function to handle user input.
- Prompt the user to enter their password using `input()`.
- Call `isValidPass()` with the input string.
- Print "valid" if `isValidPass()` returns `True`.
- Print "not valid" if `isValidPass()` returns `False`.

Flowchart



```
new_var = len(re.findall("[A-Z]",stri)) !=0
new_var1 = len(re.findall("[a-z]",stri)) !=0
# new_var1 = (re.findall("[a-z]",stri)) is not None
new_var2 = len(re.findall("[0-9]",stri)) !=0
new_var3 = len(re.findall("[_,@,$]",stri)) !=0
print(new_var)
if (new_var)  and (new_var1) and (new_var2) and (new_var3) and 8<=len(stri)<=16:
print("Valid")
else:
print("Invalid")
```

```
user_string = input("Enter a string: ")

print("Total number of characters:", len(user_string))


print("String repeated 10 times:", user_string * 10)

print("First character:", user_string[0])


print("First three characters:", user_string[:3])


print("Last three characters:", user_string[-3:])


print("String backwards:", user_string[::-1])

if len(user_string) >= 7:
    print("Seventh character:", user_string[6])
else:
    print("The string is not long enough to have a seventh character.")


print("String with first and last characters removed:", user_string[1:-1])


print("String in all caps:", user_string.upper())


print("String with 'a' replaced by 'e':", user_string.replace('a', 'e'))
```

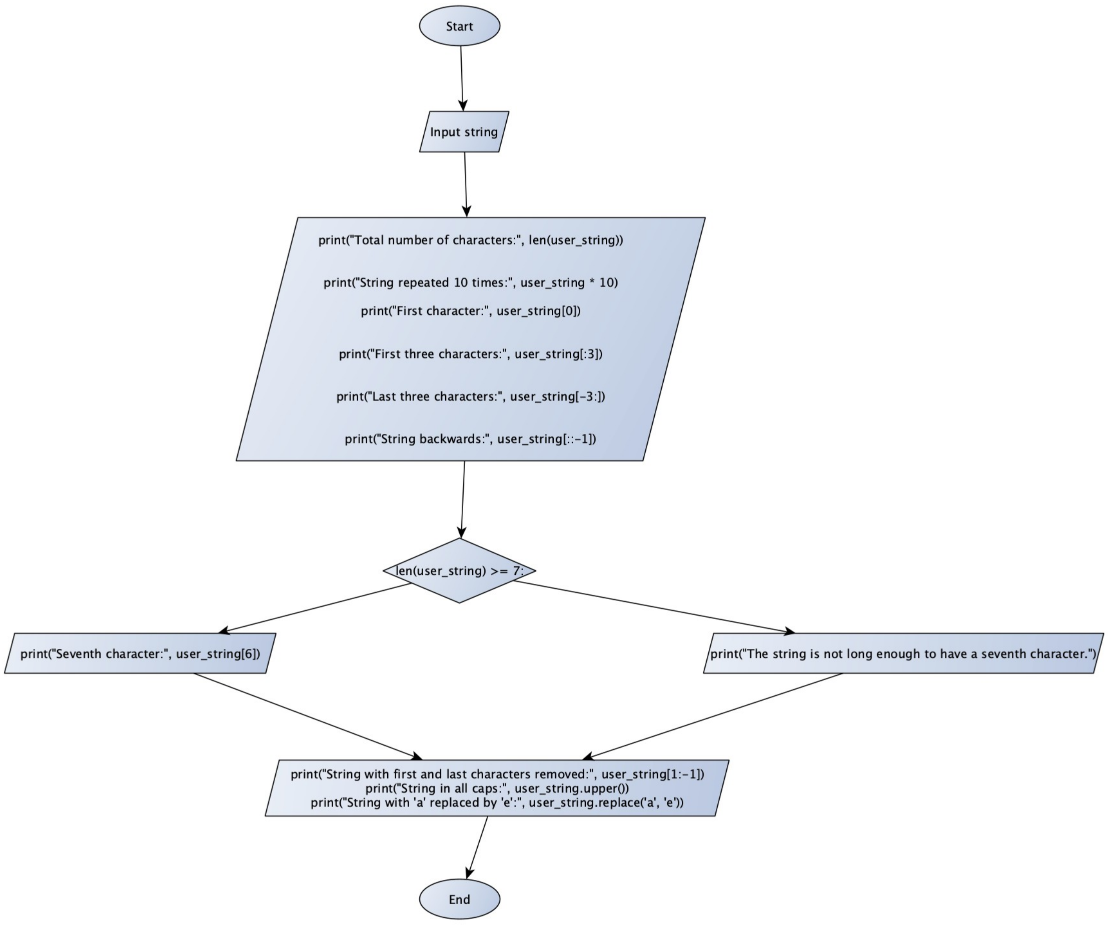
ALGO

User Input:

- Prompt the user to enter a string and store it in `user_string`.
- **Display Total Number of Characters**:
- Use `len()` to calculate and print the total number of characters in `user_string`.
- **Display String Repeated 10 Times**:
- Print `user_string` repeated 10 times using `user_string * 10`.
- **Display First Character**:
- Access and print the first character of `user_string` using `user_string[0]`.
- **Display First Three Characters**:
- Slice and print the first three characters of `user_string` using `user_string[:3]`.
- **Display Last Three Characters**:
- Slice and print the last three characters of `user_string` using `user_string[-3:]`.
- **Display String Backwards**:
- Reverse and print `user_string` using the slicing `user_string[::-1]`.
- **Display Seventh Character (if String is Long Enough)**:
- Check if the length of `user_string` is 7 or more.
    - If yes, print the seventh character using `user_string[6]`.
    - If no, print a message indicating the string is not long enough to have a seventh character.
- **Display String with First and Last Characters Removed**:

- Print `user_string` with the first and last characters removed by using `user_string[1:-1]`.
- **Display String in All Uppercase**:
- Print `user_string` converted to uppercase using `user_string.upper()`.
- **Display String with 'a' Replaced by 'e'**:
- Use `user_string.replace('a', 'e')` to replace all occurrences of 'a' with 'e' and print the result.

Flowchart


Start
Input string
print("Total number of characters:", len(user_string))
print("String repeated 10 times:", user_string * 10)
print("First character:", user_string[0])
print("First three characters:", user_string[:3])
print("Last three characters:", user_string[-3:])
print("String backwards:", user_string[::-1])
len(user_string) >= 7:
print("Seventh character:", user_string[6])
print("The string is not long enough to have a seventh character.")
print("String with first and last characters removed:", user_string[1:-1])
print("String in all caps:", user_string.upper())
print("String with 'a' replaced by 'e':", user_string.replace('a', 'e'))
End

######################

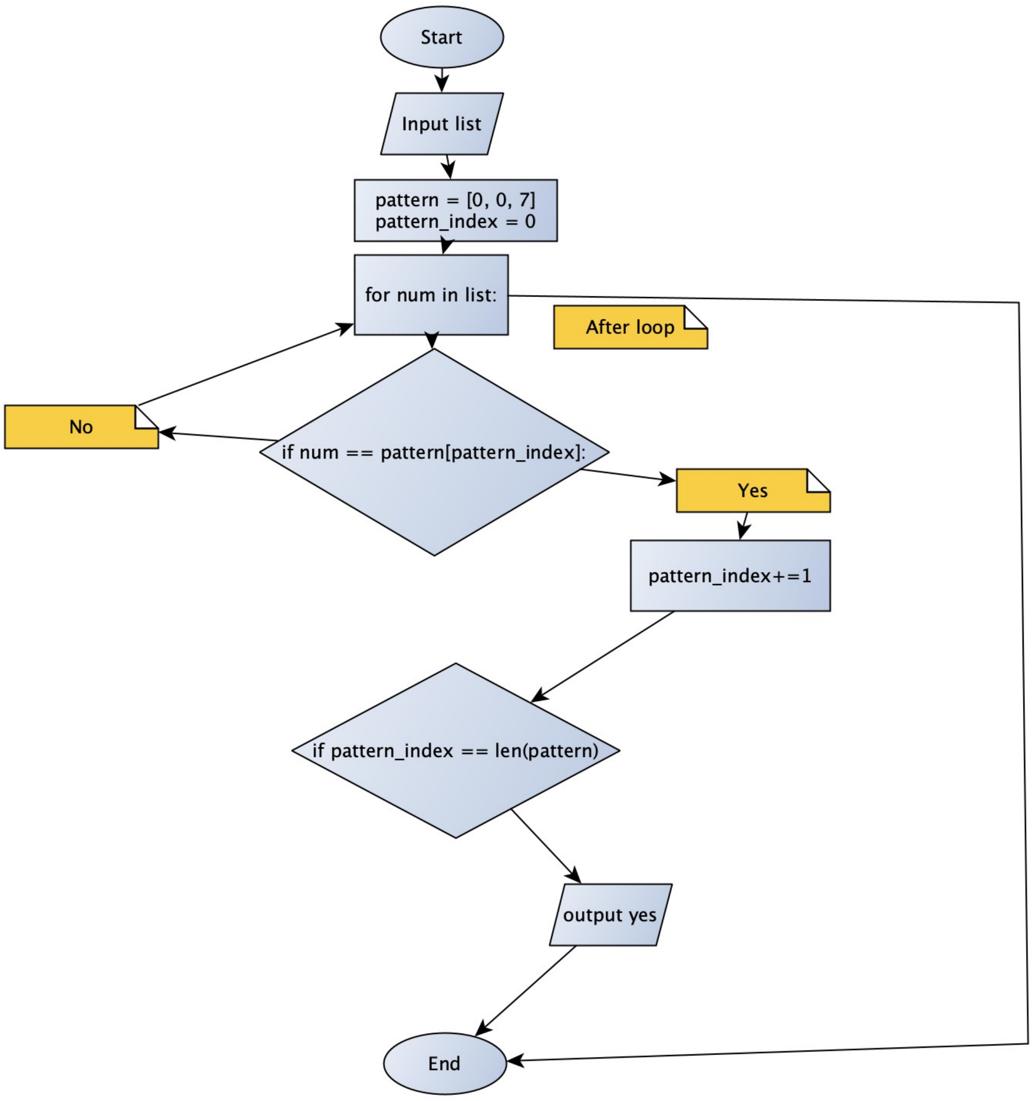
code

```
def contains_007(nums):
    pattern = [0, 0, 7]
    pattern_index = 0

    for num in nums:
        if num == pattern[pattern_index]:
            pattern_index += 1
            if pattern_index == len(pattern):
                return True
    return False

print(contains_007([1, 2, 4, 0, 0, 7, 5]))
print(contains_007([1, 0, 2, 4, 0, 5, 7]))
print(contains_007([1, 7, 2, 0, 4, 5, 0]))
```

Algorithm

Define the Target Pattern:

- Define the target pattern `[0, 0, 7]` in a list named `pattern`.
- Initialize a variable `pattern_index` to 0. This variable will track the current position in the `pattern` to check against elements in the input list.
- **Loop Through the Input List**:
- Iterate through each number `num` in the input list `nums`.
- **Check for Sequence Match**:
- For each number `num`:
  - If `num` matches the current element in the `pattern` (determined by `pattern_index`), increment `pattern_index` by 1 to move to the next element in `pattern`.
  - If `pattern_index` equals the length of `pattern`, it means all elements of `[0, 0, 7]` have been found in order. Return `True` to indicate the sequence is present.
- **Return Result**:
- If the loop completes without finding the full sequence `[0, 0, 7]`, return `False`.
- **Test Cases**:
- Call the function `contains_007()` with various lists of numbers to check if they contain the sequence `[0, 0, 7]` in the correct order.



```
###############

def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

def twin_primes(limit):
    for n in range(3, limit, 2):
        if is_prime(n) and is_prime(n + 2):
            print(f"{n},{n + 2} - Twin Prime")

twin_primes(1000)
```

Algorithm

Define Prime Check Function (`is_prime(n)`):

- If `n` is less than or equal to 1, return `False` (since prime numbers are greater than 1).
- Otherwise, for each integer `i` from 2 up to the square root of `n` (rounded up), check if `n` is divisible by `i`.
  - If `n` is divisible by any `i`, return `False` (indicating `n` is not a prime).
- If no divisors are found, return `True` (indicating `n` is prime).

- **Define Twin Primes Function** (`twin_primes(limit)`):

- Iterate through odd numbers `n` from 3 up to the specified `limit`.
  - This starts at 3 and increments by 2 to skip even numbers (only odd numbers can be twin primes when the second number is `n + 2`).
- For each odd number `n`, check if both `n` and `n + 2` are prime numbers by calling `is_prime(n)` and `is_prime(n + 2)`:
  - If both are prime, print the pair as a twin prime in the format `"{n},{n + 2} - Twin Prime"`.

- **Run the Twin Primes Function**:

- Call `twin_primes(1000)` to find and print all twin primes up to 1000.

###############

Code

```
def is_pangram(sentence):
    alphabet = set("qwertyuiopasdfghjklzxcvbnm")
    return alphabet.issubset(set(sentence.lower()))

print(is_pangram("The quick brown fox jumps over the lazy dog"))
```

Algorithm

Define Pangram Check Function (`is_pangram(sentence)`):

- Create a set `alphabet` containing all the lowercase letters of the English alphabet: `{‘qwertyuiopasdfghjklzxcvbnm’}`.
- Convert the input `sentence` to lowercase and create a set of characters from it. This set will contain each unique character in the sentence, regardless of case.

- **Check for Pangram**:

- Use the `issubset()` method to check if `alphabet` is a subset of the characters in `sentence`:
  - If all alphabet letters are found in `sentence`, return `True` (indicating `sentence` is a pangram).
  - If any alphabet letter is missing, return `False`.

- **Test the Function**:

- Call `is_pangram()` with a sample sentence, such as `"The quick brown fox jumps over the lazy dog"`, to verify if it returns `True`.

```
###############
import re

def extract_emails(filename):
    with open(filename, 'r') as file:
        content = file.read()
    return re.findall(r'[a-zA-Z0-9.+]+@[a-zA-Z0-9.-]+\.[a-zA-Z]', content)

# Test case
emails = extract_emails('textfile.txt')
print(emails)
```

Define Email Extraction Function (`extract_emails(filename)`):

- Open the file with the specified `filename` in read mode.
- Read the entire content of the file into a string variable `content`.
- **Use Regular Expression to Find Emails**:
- Use `re.findall()` with the regular expression pattern `r'[a-zA-Z0-9.+]+@[a-zA-Z0-9.-]+\.[a-zA-Z]'` to search for all email addresses in `content`.
    - `[a-zA-Z0-9.+]+`: Matches one or more alphanumeric characters or `.` or `+` symbols before the `@`.
    - `@[a-zA-Z0-9.-]+`: Matches the domain part, allowing letters, numbers, dots, and hyphens.
    - `\.[a-zA-Z]`: Matches the top-level domain, allowing letters after a dot.
- **Return Extracted Emails**:
- Return the list of matched email addresses from `re.findall()`.
- **Test the Function**:
- Call `extract_emails('textfile.txt')` to extract email addresses from `textfile.txt`.
- Print the list of extracted email addresses.

```
######
f = open("text","r")
z  = f.read()
y = z.split("\n")
cout = 0
for x in y:
    cout += len(x.split(" "))
# print chars line and words
print(len(z)) # chAracters
print(len(y)) # lines
print(cout)
```

Open and Read the File:

- Open the file `"text"` in read mode using `open("text", "r")` and assign the file object to `f`.
- Read the entire content of the file into the variable `z` using `f.read()`.

2. **Split Content into Lines**:
    - Split the content `z` into individual lines using `split("\n")` and store the result in `y`. Each element of `y` represents a line from the file.
3. **Count the Number of Words**:
    - Initialize a counter variable `cout` to 0.
    - Iterate over each line `x` in the list `y`.
        - For each line, split it into words using `split(" ")` and count the number of words. The `split(" ")` method splits the line into words by spaces.
        - Add the number of words in each line to `cout`.
4. **Print Results**:
    - Print the number of characters in the file using `len(z)` which gives the length of the entire content of the file.
    - Print the number of lines using `len(y)` which gives the number of lines (the length of the list `y`).
    - Print the total number of words using `cout`.

Summary of the Output:

- `len(z)` gives the number of characters in the file (including spaces, newline characters, etc.).
- `len(y)` gives the number of lines in the file.
- `cout` gives the total number of words in the file.

```
#######
def count_word_occurrences(filename, search_word):
    with open(filename, 'r') as file:
        content = file.read().lower()
    return content.split().count(search_word.lower())

# Text file should be created in moodle
word_count = count_word_occurrences('textfile.txt', 'specific_word')
print(f"Occurrences of 'specific_word': {word_count}")
```

Define Word Count Function (`count_word_occurrences(filename, search_word)`):

- Open the file with the specified `filename` in read mode (`'r'`), and assign the file object to `file`.

- Read the entire content of the file into a variable `content` using `file.read()`.
- Convert `content` to lowercase using `.lower()` to make the search case-insensitive.

- **Split Content into Words**:

- Split the content into individual words using `.split()`. This splits the string at any whitespace and returns a list of words.

- **Count Occurrences of the Search Word**:

- Convert the `search_word` to lowercase using `search_word.lower()` to ensure the search is case-insensitive.
- Use the `.count()` method to count how many times the lowercase `search_word` appears in the list of words (the result of `content.split()`).

- **Return the Count**:

- Return the count of occurrences.

- **Test the Function**:

- Call the `count_word_occurrences()` function with the filename `'textfile.txt'` and the word `'specific_word'`.
- Print the result, which will display the number of occurrences of `'specific_word'` in the text file.