

**INDEX**



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Exp. No.** | **Date** | **List of Experiments** | **Aim & Algorith m(10)** | **Imple mentat**  **ion (20)** | **Output and**  **Viva (20)** | **Total Mark**  **s (50)** | **Signatur e** |
| **1.** |  | Count number of characters, words and lines in a file |  |  |  |  |  |
| **2.** |  | Count the number of vowels and consonants in a file |  |  |  |  |  |
| **3.** |  | String Operations(any ten without library functions) |  |  |  |  |  |
| **4.** |  | Token Separation using high level language |  |  |  |  |  |
| **5.** |  | Study of LEX Tool |  |  |  |  |  |
| **6.** | LEX Programs | |  |  |  |  |  |
| **a.** |  | Count of positive and negative numbers |  |  |  |  |  |
| **b.** |  | Count of number of words, characters and lines |  |  |  |  |  |
| **c.** |  | Count of vowels and consonants |  |  |  |  |  |
| **d.** |  | Odd or Even |  |  |  |  |  |
| **e.** |  | Email Validation |  |  |  |  |  |
| **f.** |  | Validation of date |  |  |  |  |  |
| **g.** |  | String starts with ‘a’ |  |  |  |  |  |
| **h.** |  | Mobile number validation |  |  |  |  |  |
| **i.** |  | Lexical analyzer (Token Separation) using LEX |  |  |  |  |  |
| **7.** |  | Study of YACC Tool |  |  |  |  |  |
| **8.** |  | Parser program using YACC |  |  |  |  |  |
| **9.** |  | Calculator Program using LEX and YACC |  |  |  |  |  |
| **10.** |  | Three Address Code Generation using LEX and YACC |  |  |  |  |  |
| **11.** |  | Code Optimizer |  |  |  |  |  |
| **12.** |  | Code Generation |  |  |  |  |  |

**Average:**

# Ex. No : 01 Count number of characters, words and lines in a file Date:

**Aim:**

Python program to read a text file and then calculate and display the counts of words, lines, and characters presentin that file.

# Procedure:

* Initialize variables: Initialize words, lines, and characters variables to zero to keep track of the counts.
* Input the file name: Prompt the user to enter the name of the file they want to process.
* Open the file: Use the entered file name to open the file in read mode ('r').
* Iterate through each line: Loop through each line in the opened file.
* Split the line into words: Split the line using whitespace to get a list of words.
* Update word count: Add the length of the word list to the words count.
* Update line count: Increment the lines count by 1.
* Update character count: Update the characters count to the length of the current line.
* Close the file: After processing all lines, close the opened file.
* Display the results: Print the counts of words, lines, and characters.

# Code:

words = 0

lines = 0

characters = 0

a = input("Enter file name: ") with open(a, 'r') as file:

for l in file:

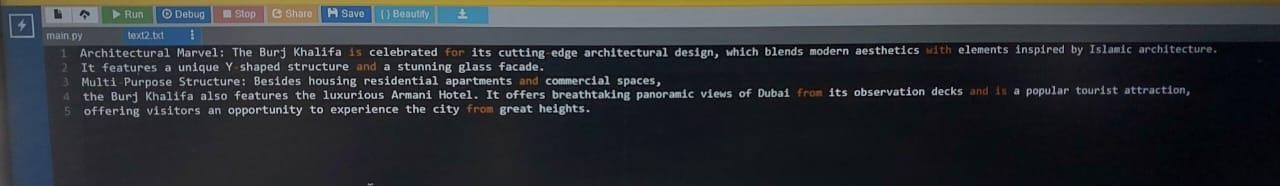
words = words + len(l.split()) lines += 1

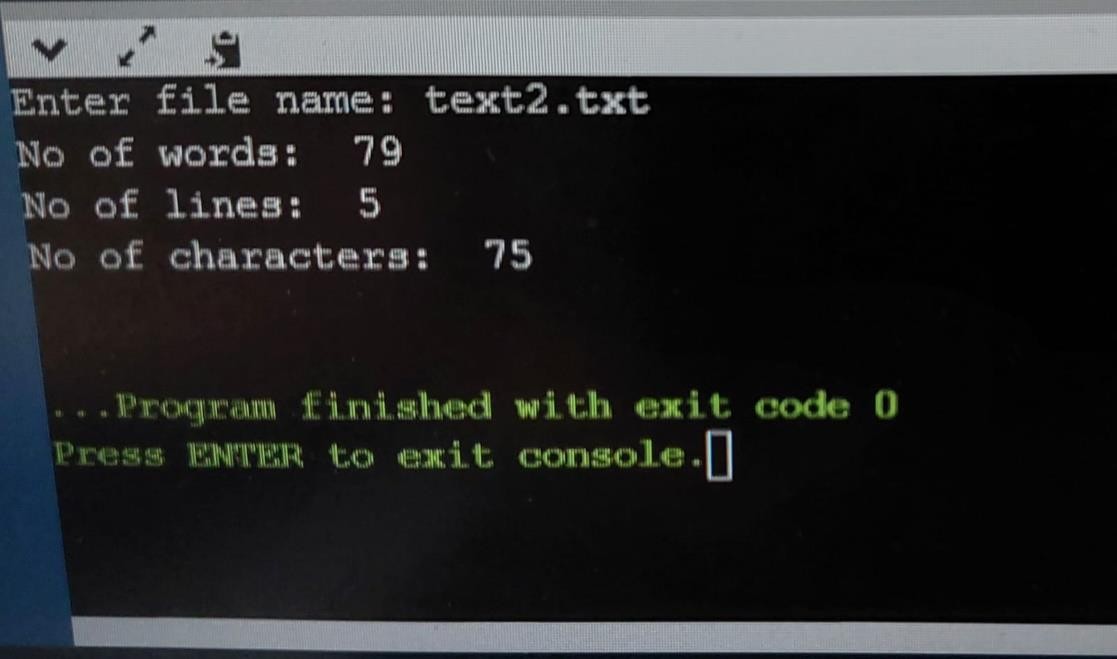
characters = len(l)

print("No of words: ",words) print("No of lines: ",lines)

print("No of characters: ",characters)

# Output:





**Result:**

The result of the code execution for the given input file would be: No of words: 79, No of lines: 5, No of characters:75.

# Ex. No : 02 Count the number of vowels and consonants in a file Date:

**Aim:**

Python program to read the content of a text file and prints the count of vowels and consonants in the text.

# Procedure:

* Define a function fun(text) that takes a string text as input.
* Initialize vowels and consonants strings containing uppercase and lowercase vowels and consonants,respectively.
* Initialize vowel\_count and consonant\_count to 0 to track the counts
* If char is found in the vowels string, increment vowel\_count by 1.
* Else if char is found in the consonants string, increment consonant\_count by 1.
* Main Program:
* Input the name of the file using filename.
* Open the specified file in read mode.
* Read the entire content of the file into the text variable.
* Call the fun function with text as an argument to get the counts of vowels and consonants.
* Print the counts of vowels and consonants.

# Code:

def fun(text):

vowels = "AEIOUaeiou"

consonants = "BCDFGHJKLMNPQRSTVWXYZbcdfghjklmnpqrstvwxyz" vowel\_count = 0

consonant\_count = 0

for char in text:

if char in vowels:

vowel\_count += 1 elif char in consonants: consonant\_count += 1

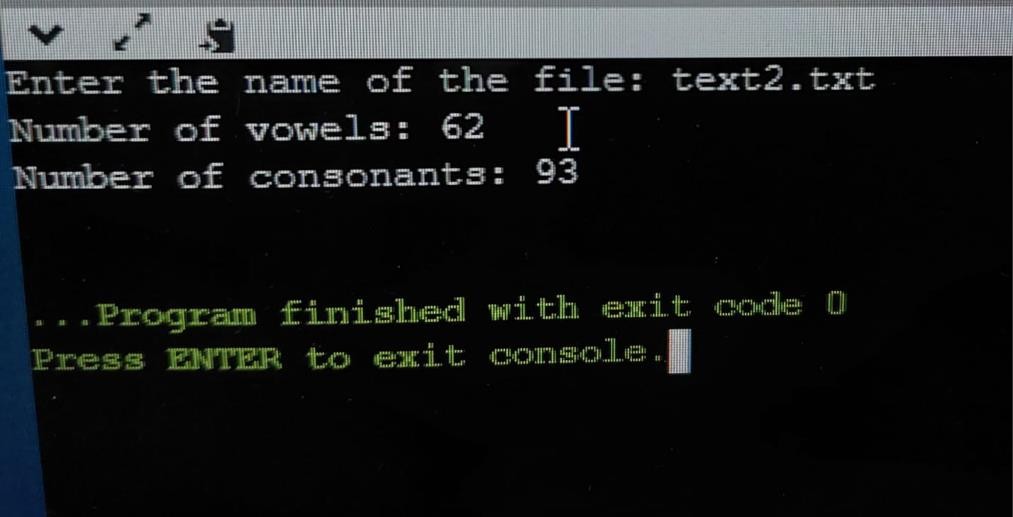
return vowel\_count, consonant\_count

filename = input("Enter the name of the file: ") with open(filename, 'r') as file:

text = file.read()

vowel\_count, consonant\_count = fun(text) print("Number of vowels:", vowel\_count) print("Number of consonants:", consonant\_count)

# Output:



**Result:**

The code reads a text file specified by the user, analyzes its content to count vowels and consonants, and then prints the counts of vowels and consonants.

# Ex. No : 03 String Operations(any ten withoutlibrary functions) Date:

**Aim:**

Python program for string operation without using inbuilt functions.

# Procedure:

## Input:

Prompt the user to input two strings, `a` and `b`.

## Length Calculation and Printing:

Initialize `count1` and `count2` to zero.

Loop through each character in string `a` and increment `count1` by 1 for each character. Loop through each character in string `b` and increment `count2` by 1 for each character.Print the length of string `a` and string `b`.

## Determine Largest String:

Compare `count1` and `count2`.

If `count1` is greater than `count2`, print `a` as the largest string; otherwise, print `b` as the largest string.

## Concatenation:

* + Initialize an empty string `res\_conc`.
  + Loop through each character in string `a` and append it to `res\_conc`.
  + Loop through each character in string `b` and append it to `res\_conc`.
  + Print the concatenated string `res\_conc`.

## Reverse the First String:

* + Initialize an empty string `revs`.
  + Loop through each character in string `a` and prepend it to `revs`.
  + Print the reversed string `revs`.

## Uppercase Conversion for First String:

* + Initialize an empty string `uppercase\_string`.
  + Loop through each character in string `a`.
  + If the character is lowercase, convert it to uppercase using ASCII manipulation and append it to

`uppercase\_string`.

* + Otherwise, append the character as is.
  + Print the uppercase converted string `uppercase\_string`.

## Lowercase Conversion for Second String:

* + Initialize an empty string `lowercase\_string`.
  + Loop through each character in string `b`.
  + If the character is uppercase, convert it to lowercase using ASCII manipulation and append it to

`lowercase\_string`.

* + Otherwise, append the character as is.
  + Print the lowercase converted string `lowercase\_string`.

## Palindrome Check for First String:

* + Initialize a boolean variable `is\_palindrome` to `True`.
  + Loop through the first half of string `a`.
  + Compare each character from the beginning with its corresponding character from the end.
  + If any mismatch is found, set `is\_palindrome` to `False` and break the loop.
  + Print whether `a` is a palindrome or not.

## Substring Extraction:

* + Given a predefined string `string` and start and end indices, initialize an empty string `substring`.
  + Loop through the characters of `string` starting from `start\_index` up to (but not including) `end\_index`.
  + Append each character to `substring`.
  + Print the extracted `substring`.

## Space Removal:

* + Given a predefined string `string`, initialize an empty string `no\_spaces`.
  + Loop through each character in `string`.
  + If the character is not a space, append it to `no\_spaces`.
  + Print the `no\_spaces` string after spaces have been removed.

## Word Count:

* + Given a predefined string `string`, initialize a variable `word\_count` to 1 (assuming at least one word exists).
  + Loop through each character in `string`.
  + If a space is encountered, increment `word\_count` by 1.
  + Print the calculated `word\_count`.

# Code:

a = input("Enter a string: ")b = input("Enter a string: ")

#1. Printing lengths of stringscount1

= 0

count2 = 0for i in a:

count1 = count1+1

for j in b:

count2 = count2+1 print("Length of ", a

," is: ", count1)

print("Length of ", b ," is: ", count2)

#2. Printing largest string if(count1>count2): print("Largest String is: ")print(a) else:

print(b)

#3. Concatenation

res\_conc = “” for char in a:

res\_conc = res\_conc+charfor char in b:

res\_conc = res\_conc+char

print(res\_conc)

#4. Reversing the stringrevs =

""

for char in a:

revs = char+revs print("Reverse of ", a

," is: ", revs)

# 5. Uppercase Conversion

uppercase\_string = "" for char in a:

if 'a' <= char <= 'z':

uppercase\_string += chr(ord(char) - 32)else: uppercase\_string += char

print("Uppercase:", uppercase\_string)

# 6. Lowercase Conversion

lowercase\_string = "" for char in b:

if 'A' <= char <= 'Z':

lowercase\_string += chr(ord(char) + 32)else: lowercase\_string += char

print("Lowercase:", lowercase\_string)

# 7. Checking Palindrome

is\_palindrome = True for i in range(count1 // 2):if a[i]

!= a[count1 - 1 - i]: is\_palindrome = False

break

print("Palindrome:", is\_palindrome)

# 8. Extracting Substringstring = "Hello, world!" start\_index = 7 end\_index = 12

substring = ""

for i in range(start\_index, end\_index):

substring += string[i]

print("Substring:", substring)

# 9. Removing Spaces string = "

Hello, world! " no\_spaces = ""

for char in string:

if char != ' ':

no\_spaces += char print("No Spaces:", no\_spaces)

# 10. Counting Words

string = "Hello, world! Welcome to Python programming." word\_count = 1

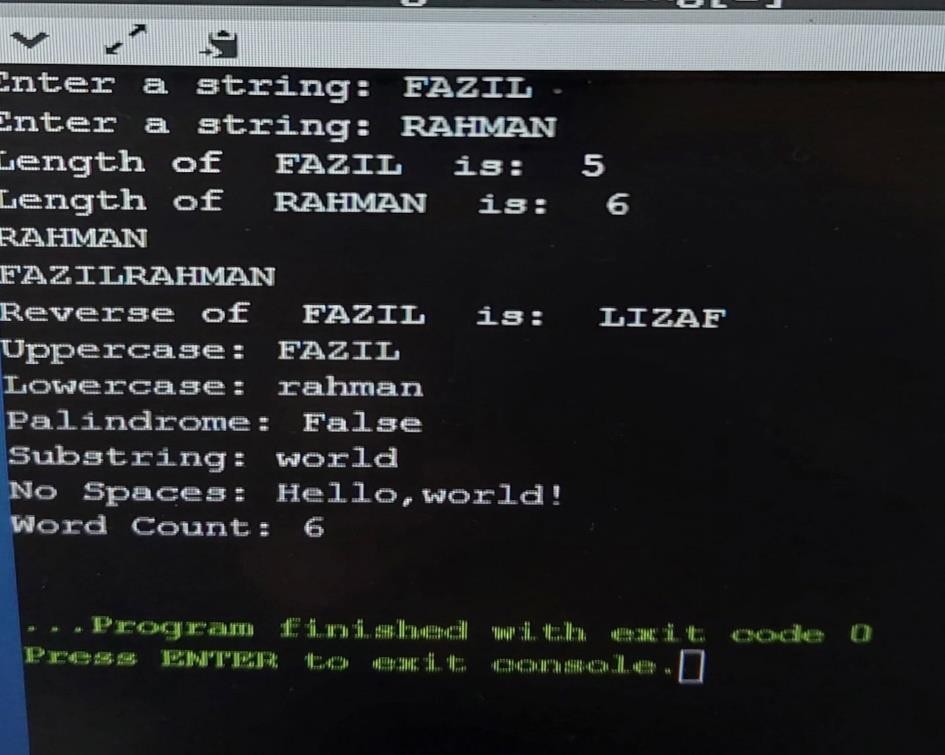
for char in string:

if char == ' ': word\_count +=

1

print("Word Count:", word\_count)

# Output:



**Result:**

Hence the string operations executed successfully without using inbuilt function.

# Ex. No : 04 Token Separation using high level language Date:

**Aim:**

To perform tokenization and classification of tokens within a script file named "Code.py."

# Procedure:

* **Open the File**
  + Open the file "code.py" for reading.
* **Define Dictionaries for Token Classification:**
  + Define dictionaries to classify tokens into their respective types, such as numbers, operators, data types, punctuation symbols, identifiers, and keywords.
* **Read the File:**
  + Read the content of the file into a variable.
* **Tokenization and Analysis:**
  + Split the file content into lines.
  + For each line:
  + Split the line into tokens based on spaces.
  + For each token:
  + Check its type using the defined dictionaries.
  + Print information about the token if it matches a dictionary key.
  + Print a separator line.

# Code:

# Open the file "Simple code.py" for reading file = open("Simple code.py")

# Define dictionaries to map tokens to their types numbers = {'6' : 'integer' , '8' : 'integer'} numbers\_keys = numbers.keys()

operators = {'=' : 'Assignment op','+' : 'Addition op','-' : 'Subtraction op','/' : 'Division op','\*' : 'Multiplication op','<' : 'Lessthan op','>' : 'Greaterthan op' }

operators\_key = operators.keys()

data\_type = {'int' : 'integer type', 'float': 'Floating point' , 'char' : 'Character type', 'long' : 'long int' } data\_type\_key = data\_type.keys()

punctuation\_symbol = { ':' : 'colon', ';' : 'semi-colon', '.' : 'dot' , ',' : 'comma' } punctuation\_symbol\_key = punctuation\_symbol.keys()

identifier = { 'a' : 'id', 'b' : 'id', 'c' : 'id' , 'd' : 'id' , '(' : 'id' ,')':'id'} identifier\_key = identifier.keys()

keywords = { 'if' : 'keyword', 'else' : 'keyword', 'print' : 'keyword' , 'for' : 'keyword' } keywords\_key = keywords.keys()

# Initialize a flag dataFlag = False

# Read the content of the file into 'a'a = file.read()

# Initialize a counter for line numbers count = 0

# Split the content of the file into lines program = a.split("\n")

# Iterate through each line of the programfor line in program:

count = count + 1

# Print the line number and the original line of code print("line:", count, "\n", line)

# Split the line into individual tokens based on spaces tokens = line.split(' ')

# Print the tokens found on the line print("Tokens are", tokens)

# Iterate through each token in the linefor token in tokens:

# Check if the token matches any keys in the dictionariesif token in operators\_key:

print("operator is ", operators[token])if token in data\_type\_key:

print("datatype is", data\_type[token])if token in punctuation\_symbol\_key:

print(token, "Punctuation symbol is", punctuation\_symbol[token])if token in identifier\_key:

print(token, "Identifier is", identifier[token])if token in numbers\_keys:

print(token, "numbers is", numbers[token])if token in keywords\_key:

print(token, " is", keywords[token])

# Reset the dataFlag (not clear how dataFlag is used in this code) dataFlag = False

# Print a separator line

print(" \_")

# Code.py:

print('I am FAZIL') f = 10

for count in range(1, 11):

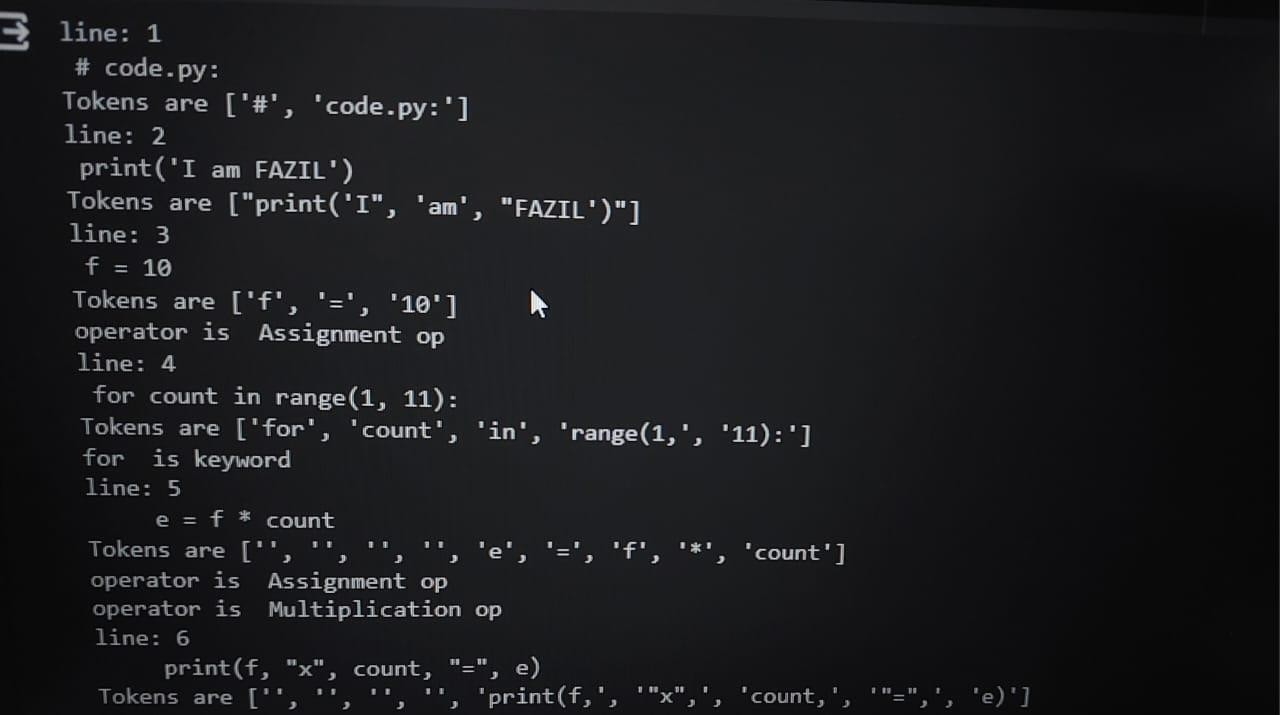
e = f \* count

print(f, "x", count, "=", e)

# Calculate and print the square of 'e' square = e \* e

print(f"The square of {e} is {square}")

# Output:



**Result:**

The above code reads a Python script file called "code.py," tokenizes its content, and categorizes the tokensinto various types such as numbers, operators, data types, punctuation symbols, identifiers, and keywords. It then prints out the token type and description for each token found in the script, line by line.