# T1A3: Terminal Application

## Text Manipulation Application

Kiran Warren 2023

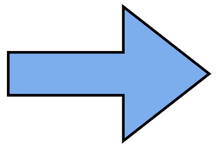
# Presentation Contents

- Introduction to Application
  - *Purpose*
  - *Functions*
- Replace Word Function
  - *Different Algorithms*
- Double Space Function
- Encryption Functions
  - *Method Chosen*
  - *Creating the Key*
- Word & Character Count
- Word Occurrence Functions

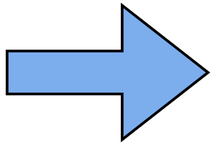
# Text Manipulation Application

The text manipulation application provides some analysis and manipulation of plain text files.

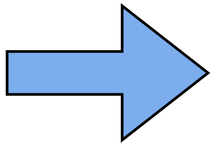

Input File

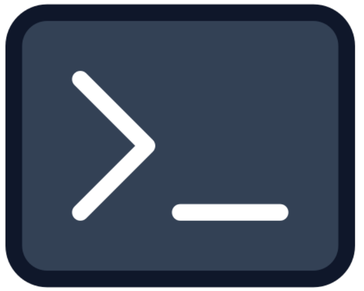


Terminal Application




Output File





Output Printed in Console

*Images references in README.md*

# Text Manipulation Application

My application provides the following functions:

## *File Manipulation*

**Replace Word Function** - Replace a target word within a text file with another word.

**Double Space Function** - Double spaces the content of a text file.

**Encrypt File Function** - Symmetrically encrypts text file contents with a user-specified password.

**Decrypt File Function** - Decrypts file encrypted by the previous function using the user's chosen password.

## *File Analysis*

**Word/Character Count** - Gives the user a count of the words and characters within a text file.

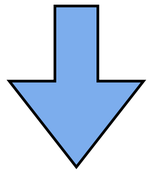
**Top Occurring Words** - Gives the user the most frequently used words within a text file, and how many times they occur.

**Word Occurrence** - Gives the user the count of how many times a particular word is used in a text file.

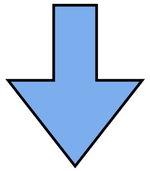
# Replace Word Function

User chooses which text file within the ‘inputs’ folder shall be imported

```
1    Hello, everyone!
2    This is a test/example file.
```



What word should be replaced? **everyone**
What word should replace “everyone”? **world**



New file created in the ‘outputs’ folder

```
1    Hello, world!
2    This is a test/example file.
```

# Replace Word Function - cont.

Main matching algorithm re-written a few times!

- **String .replace method.**

Initial idea was the import the file contents as string and use the string .replace method to replace all instances. Realised that this produced bad results!

Replace cat with dog...
=
The Very Hungry Dogerpillar

- **List of all matching variations.**

Next idea was to break up the imported string into substrings. Then match each substring with the target word. Needed a list of variations on the matching word to account for punctuation and case. Realised there were far too many variations to account for!

Cat cat CAT! cat, cat. cat? Cat?
“Cat cat” “cat” (cat cat) (cat)
= too many variations

- **Match while ignoring punctuation & case. Utilise .replace method.**

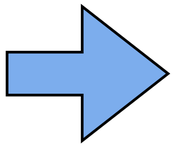
Working solution was to create a second substring list with removed punctuation and forced lower case. Easy to match with target word. Record matching indices in a set and use the string .replace method on the original list at all set values.

Encountered issue with “/“
Cat/dog considered 1 word
Created a special case

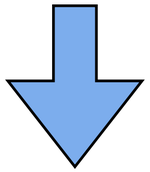
# Double Space Function

Double space function could be easily created with helper functions created while coding the replace function.

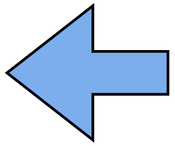
“Start with a string read from text file”



[“Split”, “into”, “list”, “of”, “substrings”]



“Join substrings together with double space delimiter”



Write to a new file.
User specifies new file name.

# Encryption Functions

Created two encryption functions; one to encrypt and the other to decrypt encrypted files.

**Spent a lot of time figuring out what method I should use to encrypt!**

- Initially wanted to encrypt data manually. Convert read string to bytes and somehow “add” to it in a reversible way.

b’Hello’ + b’random generated key’

- Looked at encoding the string in a manner that could be manipulated mathematically.

b’Hello’ -> 54 34 23 23 58

- Next I looked at simple, manual encryption techniques such as a Caesar’s Cipher.

Hello -> Ifmmp

- Finally settled on utilising cryptography library, specifically the Fernet symmetrical encryption.

from cryptography.fernet import Fernet

# Encryption Functions - cont.

Module came with a key generator, but I didn't want the user managing two files.

**Fernet object required 32-byte, base-64 encoded, URL-safe key for encryption.**

- For the simplicity of the application, I didn't want the user to manage two separate files, i.e. encrypted file and key file.

- Simplest solution for the user would be to only have to memorise a password of their choosing.

- Had difficulties getting the Fernet object to accept the keys I was passing to it. Figured out I had to follow this process: get password from user -> pad to 32 characters -> ascii encoding -> base64 encoding.

- Password given by the user could then be used to encrypt and decrypt their text file.

# Word & Character Count Function

**Word/character counts required a for-loop due to the preservation of newlines from import function.**

- **Word count:** Count number of substrings, excluding newlines (\n).

- **Character Count exc. Spaces:** Cumulative addition of all substring lengths, excluding newlines.

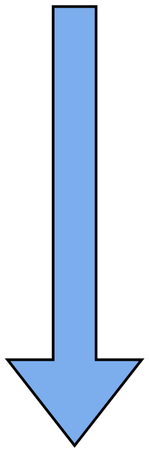
- **Character Count inc. Spaces:** Length of input string minus number of newlines.

```
Your text file has been analysed. The results are below.
Word Count: 548
Character Count (excluding spaces): 2577
Character Count (including spaces): 3115

Press enter to return to the main menu.
```

# Top Word Occurrences & Word Occurrence

Word occurrence functions logic is similar, but diverges when it comes to output.

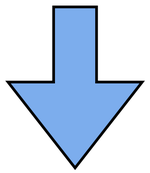


“Start with! a string (read) FROM text file”

“Remove punctuation and force lowercase”

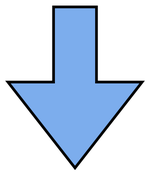
[‘split’, ‘into’, ‘list’, ‘of’, ‘substrings’]

{‘add’: 3, ‘words’: 1, ‘to’: 12, ‘dictionary’: 1}



***Specific Word***

The key we’re looking for is the user-specified word.
Print the value of the key.



***Top Words***

Get max(dict, get key) and delete key:value from dictionary.
Repeat how ever many times the user requests.

# Thank you for watching!

Kiran Warren 2023