

AOS ASSIGNMENT – 1

Implement a Pseudo Distributed Word Count Application in Python

Introduction:

The assignment includes three key tasks: firstly, creating a large text file and splitting it into smaller ones with one paragraph each; secondly, implementing word count programs using Python's threading and multiprocessing modules to efficiently process the text files; and finally, developing a master process to coordinate word count aggregation and generating a comprehensive report of word frequencies across all files.

Code and Implementation of Tasks:

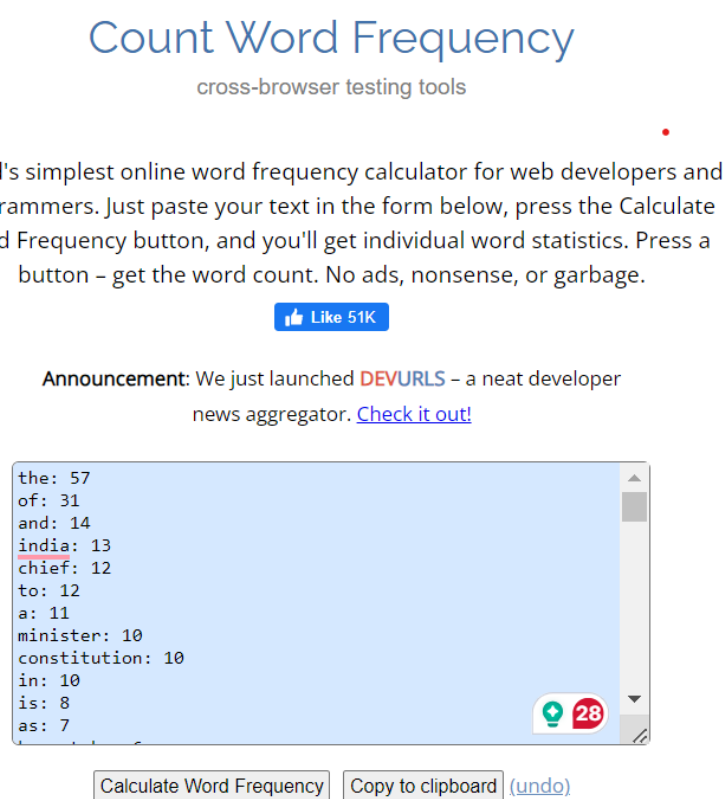
Task 1: Data splitting

Task 1.1:

A large text file is created and named “large.txt”. Have included 8 paragraphs.

Task 1.2:

Found word count in the website <https://www.browserling.com/tools/word-frequency> and the output screenshots are



The screenshot shows the 'Count Word Frequency' website interface. At the top, the title 'Count Word Frequency' is displayed in a large blue font, with the subtitle 'cross-browser testing tools' in a smaller grey font below it. A red dot is visible to the right of the subtitle. The main text area contains the following description: 'World's simplest online word frequency calculator for web developers and programmers. Just paste your text in the form below, press the Calculate Word Frequency button, and you'll get individual word statistics. Press a button – get the word count. No ads, nonsense, or garbage.' Below this text is a blue button with a thumbs-up icon and the text 'Like 51K'. An announcement banner follows, stating: 'Announcement: We just launched DEVURLS – a neat developer news aggregator. Check it out!'. The central part of the screenshot is a large light blue box containing a list of words and their frequencies: 'the: 57', 'of: 31', 'and: 14', 'india: 13', 'chief: 12', 'to: 12', 'a: 11', 'minister: 10', 'constitution: 10', 'in: 10', 'is: 8', and 'as: 7'. In the bottom right corner of this box is a green speech bubble icon with a red circle containing the number '28'. At the bottom of the screenshot are two buttons: 'Calculate Word Frequency' and 'Copy to clipboard', followed by a blue '(undo)' link.

Count Word Frequency
cross-browser testing tools

World's simplest online word frequency calculator for web developers and programmers. Just paste your text in the form below, press the Calculate Word Frequency button, and you'll get individual word statistics. Press a button – get the word count. No ads, nonsense, or garbage.

Like 51K

Announcement: We just launched DEVURLS – a neat developer news aggregator. Check it out!

```
the: 57
of: 31
and: 14
india: 13
chief: 12
to: 12
a: 11
minister: 10
constitution: 10
in: 10
is: 8
as: 7
```

Calculate Word Frequency Copy to clipboard (undo)

Count Word Frequency

cross-browser testing tools

World's simplest online word frequency calculator for web developers and programmers. Just paste your text in the form below, press the Calculate Word Frequency button, and you'll get individual word statistics. Press a button – get the word count. No ads, nonsense, or garbage.

👍 Like 51K

Announcement: We just launched [DEVURLS](#) – a neat developer news aggregator. [Check it out!](#)

```
karnataka: 6
assembly: 6
on: 6
was: 5
its: 5
government: 4
state: 4
with: 4
republic: 4
by: 4
it: 4
parliament: 4
delhi: 2
```

Calculate Word Frequency Copy to clipboard [\(undo\)](#)

Want to find the number of words in text?
button – get the word count. No ads, nonsense, or garbage.

👍 Like 51K

Announcement: We just launched [DEVURLS](#) – a neat developer news aggregator. [Check it out!](#)

```
promote: 1
fraternity: 1
original: 1
preserved: 1
nitrogen: 1
filled: 1
case: 1
at: 1
house: 1
new: 1
delhi: 1
```

Calculate Word Frequency Copy to clipboard [\(undo\)](#)

Want to find the number of words in text?
Use the [Word counter tool!](#)

There are too many words in the large text file because there total of 8 paragraphs, so there are too many individual words in the word count and have included only some screenshots.

Task 1.3:

The “large.txt” is divided into small text files based on the new line character and the “paragraph_into_file” is defined to set new unique name for each small text using uuid module.

```
task_13.py
task_13.py > ...
1  # Implement a Python script to split the text file into multiple (at least two) small
2  # text files. Each small text file should have exactly one paragraph. Name each small text
3  # file with a unique name.
4
5  import multiprocessing
6  import uuid
7
8  large_file_name = "large.txt"
9  paragraphs = open(large_file_name).read()
10 paragraphs = paragraphs.split("\n\n")
11
12
13 # smalls folder must exist before executing this script.
14 def paragraph_into_file(paragraph, destination="smalls/"):
15     filename = destination + uuid.uuid4().hex
16     print(paragraph, file=open(filename, "w"))
17
18
19 if __name__ == "__main__":
20     pool = multiprocessing.Pool(processes=len(paragraphs))
21     pool.map(paragraph_into_file, paragraphs)
22     pool.close()
23     pool.join()
24
```

Fig: Task 1.3

The below figure represents the small text files and its unique names, These are assigned based on the uuid module and This creates a universally unique identifier (UUID) and converts it to a hexadecimal string.

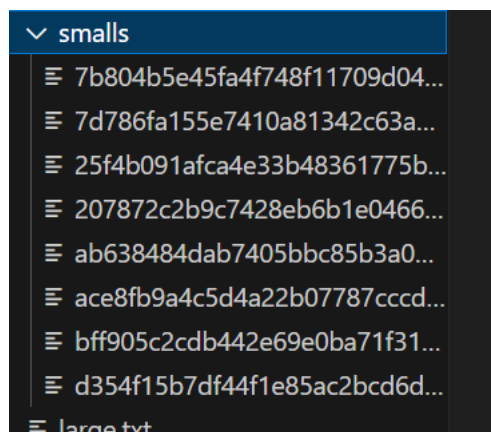


Fig : Task 1.3 Screenshots

- The Large text file will split into multiple small text files only when the task_1_3.py is compiled. So, when the task_1_3.py is compiled multiple times, more small files will be created than required. So, the task_1_3.py must be compiled only once.

utils.py

This code defines two utility functions for working with small text files stored in a directory named "smalls/". The `getSmallFileNames` function retrieves the names of files in the directory, while the `getSmallFilePaths` function constructs their full paths. Both functions ensure that the specified directory exists and return lists of file names or paths, respectively, or an empty list if the directory doesn't exist.

```

utils.py > ...
import os

smallfiles_basepath = "smallfs/"

def getSmallFileNames(smallfiles_basepath=smallfiles_basepath):
    if not os.path.exists(smallfiles_basepath):
        return []
    if not os.path.isdir(smallfiles_basepath):
        return []
    return os.listdir(smallfiles_basepath)

def getSmallFilePaths(smallfiles_basepath=smallfiles_basepath):
    smallfile_names = getSmallFileNames()
    smallfilepaths = [smallfiles_basepath + filename for filename in smallfile_names]
    return smallfilepaths

```

Task 2: Distributed word count

Task 2.1:

This Python script counts the frequency of each word in multiple small text files by processing them concurrently using threading. It defines the `getFreqCount` function, which counts the occurrences of a word in a paragraph. After reading a text file, each thread calls this function and outputs the word count result. Finally, after every thread has finished executing, it prints a message.

```

task_21.py > ...
import threading

from utils import getSmallFilePaths

def getFreqCount(paragraph, thread_name=None):
    countDict = dict()
    tokens = paragraph.split()

    for token in tokens:
        token = token.lower().strip('.,!()?[]{}":;')

        if not token:
            continue

        countDict[token] = countDict.get(token, 0) + 1

    if thread_name is None:
        return countDict

    thread_result = {"thread_name": thread_name, "thread_result": countDict}
    print(thread_result)

filepaths = getSmallFilePaths()
threads = []
for index, filepath in enumerate(filepaths, start=1):
    para = open(filepath).read()

    thread_name = f"thread {index}"
    thread = threading.Thread(target=getFreqCount, args=[para, thread_name])

```

Fig : Task 2.1

Below is the output of the task 2.1 and returns a dictionary of word counts for each small text file.

```
{'thread_name': 'thread 1', 'thread_result': {'the': 3, 'constitution': 2, 'declares': 1, 'india': 1, 'a': 2, 'sovereign': 1, 'socialist': 1, 'secular': 1, 'and': 3, 'democratic': 1, 'republic': 1, 'assures': 1, 'its': 1, 'citizens': 1, 'justice': 1, 'equality': 1, 'liberty': 1, 'endeavours': 1, 'to': 1, 'promote': 1, 'fraternity': 1, 'original': 1, '1950': 1, 'is': 1, 'preserved': 1, 'in': 2, 'nitrogen': 1, 'filled': 1, 'case': 1, 'at': 1, 'parliament': 1, 'house': 1, 'new': 1, 'delhi': 1}}
{'thread_name': 'thread 2', 'thread_result': {'following': 1, 'elections': 1, 'to': 4, 'the': 11, 'karnataka': 1, 'legislative': 1, 'assembly': 4, 'governor': 2, 'usually': 1, 'invites': 1, 'political': 2, 'party': 1, 'or': 1, 'a': 2, 'coalition': 1, 'of': 4, 'parties': 1, 'with': 1, 'majority': 1, 'seats': 1, 'form': 1, 'government': 1, 'in': 1, 'state': 1, 'appoints': 1, 'chief': 2, 'minister': 1, 'whose': 1, 'council': 1, 'ministers': 2, 'is': 3, 'collectively': 1, 'responsible': 1, 'given': 1, 'that': 1, 'he': 1, 'she': 1, 'has': 1, 'confidence': 1, 'term': 2, 'for': 1, 'five': 1, 'years': 1, 'renewable': 1, 'and': 1, 'subject': 1, 'no': 1, 'limits': 1}}
{'thread_name': 'thread 3', 'thread_result': {'it': 2, 'imparts': 1, 'constitutional': 1, 'supremacy': 2, 'not': 1, 'parliamentary': 1, 'since': 1, 'was': 2, 'created': 1, 'by': 2, 'a': 2, 'constituent': 1, 'assembly': 1, 'rather': 1, 'than': 1, 'parliament': 2, 'and': 1, 'adopted': 1, 'its': 2, 'people': 1, 'with': 1, 'declaration': 1, 'in': 1, 'preamble': 1, 'cannot': 1, 'override': 1, 'the': 1, 'constitution': 1}}
{'thread_name': 'thread 4', 'thread_result': {'chief': 3, 'minister': 2, 'of': 6, 'karnataka': 3, 'is': 2, 'the': 7, 'executive': 2, 'officer': 1, 'government': 1, 'indian': 2, 'state': 1, 'as': 1, 'per': 1, 'constitution': 1, 'india': 1, 'governor': 1, 'states': 2, 'de': 2, 'jure': 1, 'head': 1, 'but': 1, 'facto': 1, 'authority': 1, 'rests': 1, 'with': 1, 'a': 1, 'template': 1, 'applicable': 1, 'to': 1, 'all': 1, 'other': 1}}
{'thread_name': 'thread 5', 'thread_result': {'ambedkar': 1, 'and': 3, 'constitution': 3, 'of': 7, 'india': 7, 'on': 4, 'a': 1, '2015': 1, 'postage': 1, 'stamp': 1, 'it': 1, 'was': 1, 'adopted': 1, 'by': 1, 'the': 7, 'constituent': 1, 'assembly': 1, '26': 3, 'november': 1, '1949': 1, 'became': 2, 'effective': 1, 'january': 2, '1950': 1, 'replaced': 1, 'government': 1, 'act': 1, '1935': 1, 'as': 2, 'countrys': 1, 'fundamental': 1, 'governing': 1, 'document': 1, 'dominion': 1, 'republic': 2, 'to': 1, 'ensure': 1, 'constitutional': 1, 'autochthony': 1, 'its': 2, 'framers': 1, 'repealed': 1, 'prior': 1, 'acts': 1, 'british': 1, 'parliament': 1, 'in': 1, 'article': 1, '395': 1, 'celebrates': 1, 'day': 1}}
{'thread_name': 'thread 6', 'thread_result': {'the': 8, 'constitution': 2, 'of': 4, 'india': 2, 'is': 2, 'supreme': 1, 'law': 1, 'document': 1, 'lays': 1, 'down': 1, 'framework': 1, 'that': 1, 'demarcates': 1, 'fundamental': 2, 'political': 1, 'code': 1, 'structure': 1, 'procedures': 1, 'powers': 1, 'and': 3, 'duties': 2, 'government': 1, 'institutions': 1, 'sets': 1, 'out': 1, 'rights': 1, 'directive': 1, 'principles': 1, 'citizens': 1, 'based': 1, 'on': 1, 'proposal': 1, 'suggested': 1, 'by': 1, 'm': 1, 'n': 1, 'roy': 1, 'it': 1, 'longest': 1, 'written': 1, 'national': 1, 'in': 1, 'world': 1}}
{'thread_name': 'thread 7', 'thread_result': {'one': 2, 'chief': 5, 'minister': 6, 'bheave': 1, 'gowda': 1, 'went': 1, 'on': 1}}
```

Fig : Task 2.1 output

Task 2.2:

This Python script modifies the program to utilize the multiprocessing module for concurrent processing. It separates the text files into batches, and runs a different process on each batch. Several threads are used in each process to handle different files at the same time. Each thread's output is combined using a shared dictionary and locked to prevent synchronization. Finally, a multiprocessing pool is created to execute the processing function for each batch.

```
def getFreqCount(filepath, freqRef, lock):
    if freqRef is not None:
        result = freqRef
    else:
        result = dict()

    def addToken(token):
        getCleanToken = lambda token: token.lower().strip('.,!()?[]{}";: ')

        ctoken = getCleanToken(token)
        if not ctoken:
            return

        lock.acquire()
        # print(f"ACQUIRED : {threading.current_thread().name}")
        result[ctoken] = result.get(ctoken, 0) + 1
        # print(f"RELEASED : {threading.current_thread().name}")
        lock.release()

    paragraph = open(filepath).read()
    tokens = paragraph.split()
    [addToken(token) for token in tokens]
    print(result)
    return result
```

Fig : Task 2.2

Below is the output of the task 2.2

```
{'the': 3, 'constitution': 2, 'declares': 1, 'india': 1, 'a': 2, 'sovereign': 1, 'socialist': 1, 'secular': 1, 'and': 3, 'democratic': 1, 'republic': 1, 'assures': 1, 'its': 1, 'citizens': 1, 'justice': 1, 'equality': 1, 'liberty': 1, 'endeavours': 1, 'to': 1, 'promote': 1, 'fraternity': 1, 'original': 1, '1950': 1, 'is': 1, 'preserved': 1, 'in': 2, 'nitrogen': 1, 'filled': 1, 'case': 1, 'at': 1, 'parliament': 1, 'house': 1, 'new': 1, 'delhi': 1}
{'the': 14, 'constitution': 2, 'declares': 1, 'india': 1, 'a': 4, 'sovereign': 1, 'socialist': 1, 'secular': 1, 'and': 4, 'democratic': 1, 'republic': 1, 'assures': 1, 'its': 1, 'citizens': 1, 'justice': 1, 'equality': 1, 'liberty': 1, 'endeavours': 1, 'to': 5, 'promote': 1, 'fraternity': 1, 'original': 1, '1950': 1, 'is': 4, 'preserved': 1, 'in': 3, 'nitrogen': 1, 'filled': 1, 'case': 1, 'at': 1, 'parliament': 1, 'house': 1, 'new': 1, 'delhi': 1, 'following': 1, 'elections': 1, 'karnataka': 1, 'legislative': 1, 'assembly': 4, 'governor': 2, 'usually': 1, 'invites': 1, 'political': 2, 'party': 1, 'or': 1, 'coalition': 1, 'of': 4, 'parties': 1, 'with': 1, 'majority': 1, 'seats': 1, 'form': 1, 'government': 1, 'state': 1, 'appoints': 1, 'chief': 2, 'minister': 1, 'whose': 1, 'council': 1, 'ministers': 2, 'collectively': 1, 'responsible': 1, 'given': 1, 'that': 1, 'he': 1, 'she': 1, 'has': 1, 'confidence': 1, 'term': 2, 'for': 1, 'five': 1, 'years': 1, 'renewable': 1, 'subject': 1, 'no': 1, 'limits': 1}
{'the': 14, 'constitution': 2, 'declares': 1, 'india': 1, 'a': 4, 'sovereign': 1, 'socialist': 1, 'secular': 1, 'and': 4, 'democratic': 1, 'republic': 1, 'assures': 1, 'its': 1, 'citizens': 1, 'justice': 1, 'equality': 1, 'liberty': 1, 'endeavours': 1, 'to': 5, 'promote': 1, 'fraternity': 1, 'original': 1, '1950': 1, 'is': 4, 'preserved': 1, 'in': 3, 'nitrogen': 1, 'filled': 1, 'case': 1, 'at': 1, 'parliament': 1, 'house': 1, 'new': 1, 'delhi': 1, 'following': 1, 'elections': 1, 'karnataka': 1, 'legislative': 1, 'assembly': 4, 'governor': 2, 'usually': 1, 'invites': 1, 'political': 2, 'party': 1, 'or': 1, 'coalition': 1, 'of': 4, 'parties': 1, 'with': 1, 'majority': 1, 'seats': 1, 'form': 1, 'government': 1, 'state': 1, 'appoints': 1, 'chief': 2, 'minister': 1, 'whose': 1, 'council': 1, 'ministers': 2, 'collectively': 1, 'responsible': 1, 'given': 1, 'that': 1, 'he': 1, 'she': 1, 'has': 1, 'confidence': 1, 'term': 2, 'for': 1, 'five': 1, 'years': 1, 'renewable': 1, 'subject': 1, 'no': 1, 'limits': 1}
-----
{'it': 2, 'imparts': 1, 'constitutional': 1, 'supremacy': 2, 'not': 1, 'parliamentary': 1, 'since': 1, 'was': 2, 'created': 1, 'by': 2, 'a': 2, 'constituent': 1, 'assembly': 1, 'rather': 1, 'than': 1, 'parliament': 2, 'and': 1, 'adopted': 1, 'its': 2, 'people': 1, 'with': 1, 'declaration': 1, 'in': 1, 'preamble': 1, 'cannot': 1, 'override': 1, 'the': 1, 'constitution': 2}
```

Fig : Task 2.2 output

Task 3: Aggregation and Reporting

Task 3.1:

The master process for coordinating word count aggregation is implemented by this Python script. File names are retrieved from a folder, and each file is processed simultaneously using multiprocessing. Using a multiprocessing manager, the word counts from all processes are combined into a shared dictionary. In the end, the word counts are displayed in frequency order that is decreasing.

```
def getFileNames(folder_name="smalls"):
    if not os.path.exists(folder_name) or not os.path.isdir(folder_name):
        return []

    file_names = os.listdir(folder_name)
    return file_names

def getFreqCount(filepath, countDict):
    paragraph = open(filepath).read()
    tokens = paragraph.split()
    for token in tokens:
        token = token.lower().strip('.,!?([]{}":;')

        if not token: continue

        countDict[token] = countDict.get(token, 0) + 1

folderpath = "smalls/"
filenames = getFileNames()
filepaths = [folderpath + filename for filename in filenames]

if __name__ == "__main__":
    with multiprocessing.Manager() as manager:
        freqResult = manager.dict()
```

Fig : Task 3.1

Below is the output of the task 3.1

```
the : 57
of : 31
and : 14
india : 13
to : 12
chief : 12
a : 11
constitution : 10
in : 10
minister : 10
is : 8
as : 7
karnataka : 6
assembly : 6
on : 6
its : 5
was : 5
republic : 4
parliament : 4
with : 4
government : 4
state : 4
it : 4
by : 4
governor : 3
political : 3
party : 3
```

Fig : Task 3.1 output

Task 3.2:

This Python script generates a report with the total count of all the words and computes word frequencies across several small text files. Iteratively going through each file, it adds word counts to a dictionary and outputs the word counts in decreasing order of frequency.

```
def updateFreqCount(filepath, countDict):
    paragraph = open(filepath).read()
    tokens = paragraph.split()
    for token in tokens:
        token = token.lower().strip('.,!()?[]{}"':;')

        if not token: continue

        countDict[token] = countDict.get(token, 0) + 1

freqResult = dict()
filepaths = getSmallFilePaths(smallfiles_basepath)
[updateFreqCount(filepath, freqResult) for filepath in filepaths]

for key in sorted(freqResult.keys(), key=lambda x: -freqResult[x]):
    print(f"{key} : {freqResult[key]}")
```

Fig : Task 3.2

Below is the output of the task 3.2

```
the : 57
of : 31
and : 14
india : 13
to : 12
chief : 12
a : 11
constitution : 10
in : 10
minister : 10
is : 8
as : 7
karnataka : 6
assembly : 6
on : 6
its : 5
was : 5
republic : 4
parliament : 4
with : 4
government : 4
state : 4
it : 4
by : 4
governor : 3
political : 3
party : 3
```

Fig: Final Output

References

1. <https://www.studytonight.com/python/python-threading-lock-object>
2. <https://stackoverflow.com/questions/2961509/python-how-to-create-a-unique-file-name>
3. <https://stackoverflow.com/questions/1697571/threading-appears-to-run-threads-sequentially>
4. <https://www.youtube.com/watch?v=IEEhzQoKtQU&t=910s>
5. https://www.youtube.com/watch?v=fKl2JW_qrso&t=1184s
6. <https://www.youtube.com/watch?v=v5u5zSYKhFs>