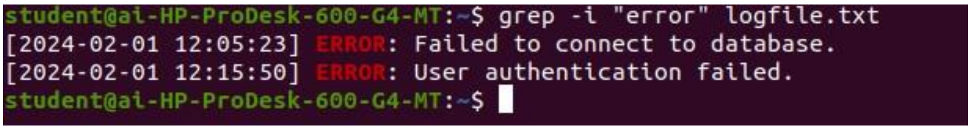
LAB TASK:423132-11/02/2025

Using `grep-TO FILTER THE TEXT(GLOBAL REG EXP PRINT)`

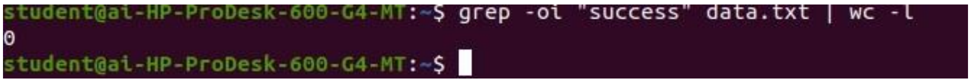
1. Find all lines containing the word **"error"** in a log file (`log.txt`).



```
student@ai-HP-ProDesk-600-G4-MT:~$ grep -i "error" logfile.txt
[2024-02-01 12:05:23] ERROR: Failed to connect to database.
[2024-02-01 12:15:50] ERROR: User authentication failed.
student@ai-HP-ProDesk-600-G4-MT:~$ 
```

-i flag: for case-insensitivity

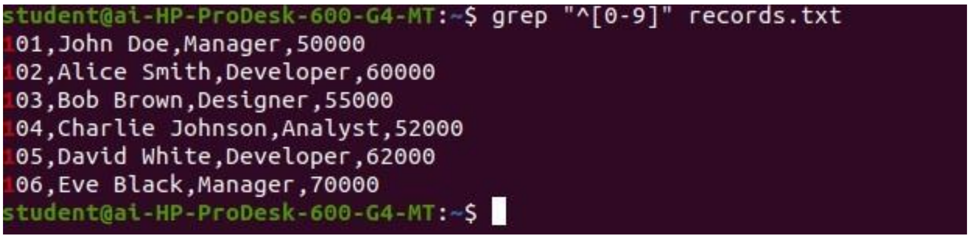
2. Count the occurrences of the word **"success"** in a file (`data.txt`).



```
student@ai-HP-ProDesk-600-G4-MT:~$ grep -oi "success" data.txt | wc -l
0
student@ai-HP-ProDesk-600-G4-MT:~$ 
```

-o flag: to print   wc -
l: to count

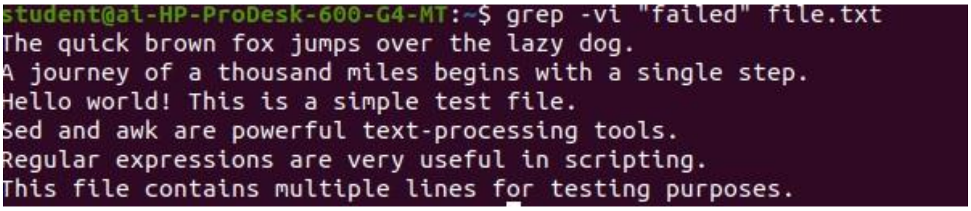
3. Extract all lines from a file (`records.txt`) that start with a digit.



```
student@ai-HP-ProDesk-600-G4-MT:~$ grep "^[0-9]" records.txt
101,John Doe,Manager,50000
102,Alice Smith,Developer,60000
103,Bob Brown,Designer,55000
104,Charlie Johnson,Analyst,52000
105,David White,Developer,62000
106,Eve Black,Manager,70000
student@ai-HP-ProDesk-600-G4-MT:~$ 
```

^:: Anchors the pattern to the start of the line.

4. Display all lines in `file.txt` that do **not** contain the word **"failed"**.

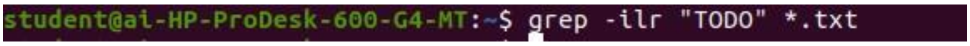


```
student@ai-HP-ProDesk-600-G4-MT:~$ grep -vi "failed" file.txt
The quick brown fox jumps over the lazy dog.
A journey of a thousand miles begins with a single step.
Hello world! This is a simple test file.
Sed and awk are powerful text-processing tools.
Regular expressions are very useful in scripting.
This file contains multiple lines for testing purposes.
```

-v : to invert the match

5. Find all `.txt` files in the current directory that contain the word **"TODO"**.

L:to list , R:Recursively searches subdirectories.

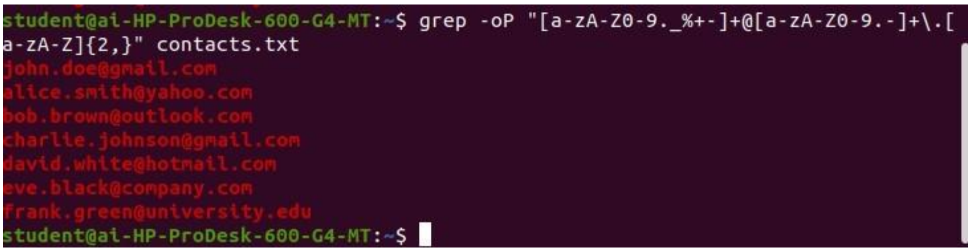


```
student@ai-HP-ProDesk-600-G4-MT:~$ grep -ilr "TODO" *.txt
```

6. Extract email addresses from `contacts.txt` (Hint: Use regex).

Regex -**Regular Expression** is used for pattern matching within strings of text.

LAB TASK:423132-11/02/2025



```
student@ai-HP-ProDesk-600-G4-MT:~$ grep -oP "[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[
a-zA-Z]{2,}" contacts.txt
john.doe@gmail.com
alice.smith@yahoo.com
bob.brown@outlook.com
charlie.johnson@gmail.com
david.white@hotmail.com
eve.black@company.com
frank.green@university.edu
student@ai-HP-ProDesk-600-G4-MT:~$
```

\ : MATCHES THE DOT

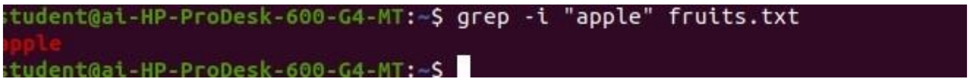
2 : HAS ATLEAST 2 CHARACTERS

//PHONE NUMBER VALIDATION

P: Uses Perl-compatible regex (needed for more advanced regex syntax).

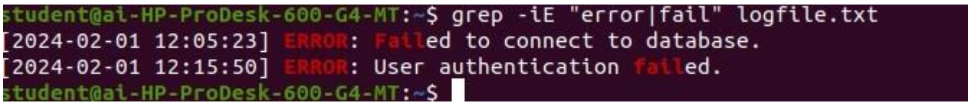
-o: Only prints the matching portion (the email addresses).

7. Find all occurrences of **"apple"**, case-insensitive, in `fruits.txt`.



```
tudent@ai-HP-ProDesk-600-G4-MT:~$ grep -i "apple" fruits.txt
pple
tudent@ai-HP-ProDesk-600-G4-MT:~$
```

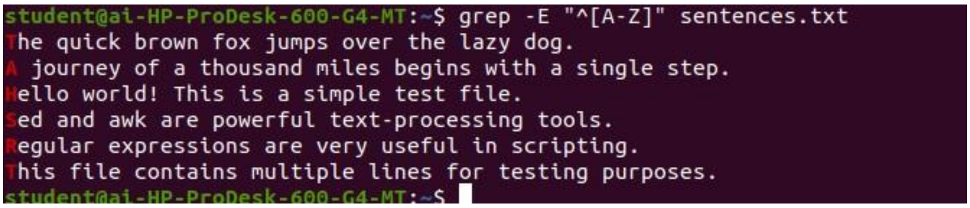
8. Find all lines in `logfile.txt` that contain **either "error" or "fail"**.



```
student@ai-HP-ProDesk-600-G4-MT:~$ grep -iE "error|fail" logfile.txt
[2024-02-01 12:05:23] ERROR: Failed to connect to database.
[2024-02-01 12:15:50] ERROR: User authentication failed.
student@ai-HP-ProDesk-600-G4-MT:~$
```

-E : to use or operator

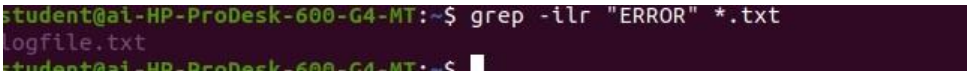
9. Display lines that **start with a capital letter** from `sentences.txt`.



```
student@ai-HP-ProDesk-600-G4-MT:~$ grep -E "^[A-Z]" sentences.txt
The quick brown fox jumps over the lazy dog.
A journey of a thousand miles begins with a single step.
Hello world! This is a simple test file.
Sed and awk are powerful text-processing tools.
Regular expressions are very useful in scripting.
This file contains multiple lines for testing purposes.
student@ai-HP-ProDesk-600-G4-MT:~$
```

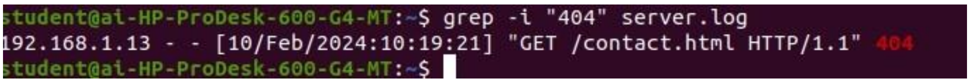
-E : for extended expns, which allows the use of more advanced regex features.

10. List only filenames from the current directory that contain the word **"project"**.(you can pick any word here that is being repeated)



```
student@ai-HP-ProDesk-600-G4-MT:~$ grep -ilr "ERROR" *.txt
logfile.txt
student@ai-HP-ProDesk-600-G4-MT:~$
```

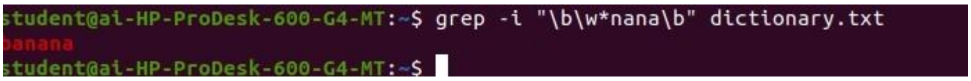
11. Find lines in `server.log` that contain **"404"**, but ignore case sensitivity.



```
student@ai-HP-ProDesk-600-G4-MT:~$ grep -i "404" server.log
192.168.1.13 - - [10/Feb/2024:10:19:21] "GET /contact.html HTTP/1.1" 404
student@ai-HP-ProDesk-600-G4-MT:~$
```

LAB TASK:423132-11/02/2025

12. Find all words in `dictionary.txt` that **end with "ing"**.

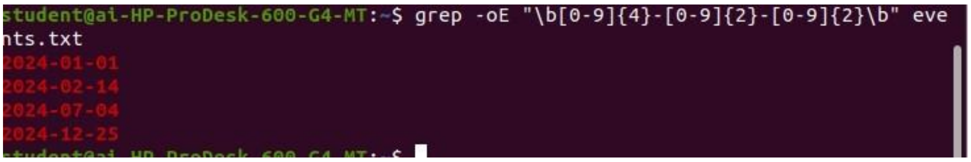


```
student@ai-HP-ProDesk-600-G4-MT:~$ grep -i "\b\w*nana\b" dictionary.txt
banana
student@ai-HP-ProDesk-600-G4-MT:~$
```

\b : boundary

\w* :Matches letters, digits, or underscores

13. Extract **dates (YYYY-MM-DD format)** from `events.txt`.

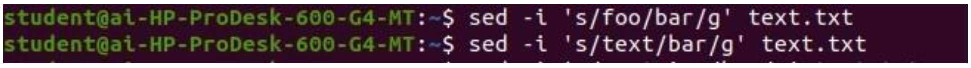


```
student@ai-HP-ProDesk-600-G4-MT:~$ grep -oE "\b[0-9]{4}-[0-9]{2}-[0-9]{2}\b" eve
nts.txt
2024-01-01
2024-02-14
2024-07-04
2024-12-25
student@ai-HP-ProDesk-600-G4-MT:~$
```

Using `sed- (stream editor)to perform operations like updation,deletion..`

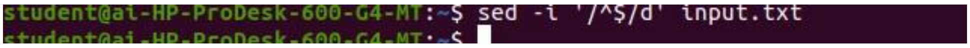
1. Replace all occurrences of **"foo"** with **"bar"** in `text.txt`.

S : substitute ,g :replace all , -i : to modify the file directly



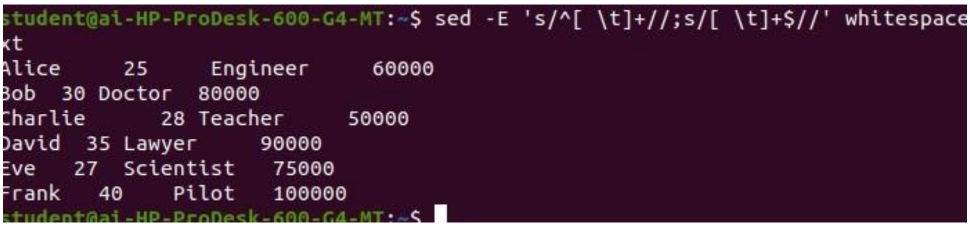
```
student@ai-HP-ProDesk-600-G4-MT:~$ sed -i 's/foo/bar/g' text.txt
student@ai-HP-ProDesk-600-G4-MT:~$ sed -i 's/text/bar/g' text.txt
```

2. Delete all blank lines from `input.txt`. ^-to match the start ,&-end



```
student@ai-HP-ProDesk-600-G4-MT:~$ sed -i '/^$/d' input.txt
student@ai-HP-ProDesk-600-G4-MT:~$
```

3. Remove leading and trailing spaces from each line in `whitespace.txt`.

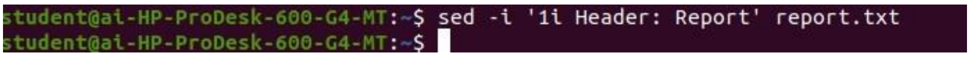


```
student@ai-HP-ProDesk-600-G4-MT:~$ sed -E 's/^[ \t]+//;s/[ \t]+$//' whitespace
xt
Alice      25      Engineer      60000
Bob  30 Doctor  80000
Charlie      28 Teacher      50000
David  35 Lawyer     90000
Eve   27  Scientist   75000
Frank   40    Pilot   100000
student@ai-HP-ProDesk-600-G4-MT:~$
```

4. Insert a new line with the text **"Header: Report"** at the beginning of `report.txt`.

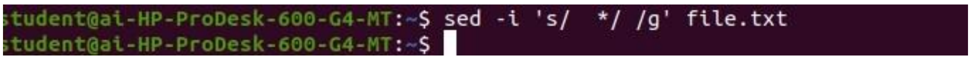
`-i` edits the file in place.

`'1i Header: Report'` inserts "`Header: Report`" at line 1.



```
student@ai-HP-ProDesk-600-G4-MT:~$ sed -i '1i Header: Report' report.txt
student@ai-HP-ProDesk-600-G4-MT:~$
```

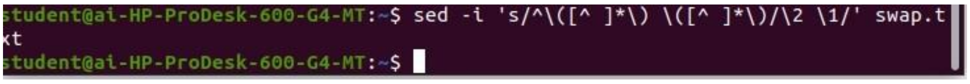
5. Replace all instances of multiple spaces with a single space in `file.txt`.



```
student@ai-HP-ProDesk-600-G4-MT:~$ sed -i 's/  */ /g' file.txt
student@ai-HP-ProDesk-600-G4-MT:~$
```

LAB TASK:423132-11/02/2025

6. Swap the first and second word in each line of `swap.txt`.



```
student@ai-HP-ProDesk-600-G4-MT:~$ sed -i 's/^\([^ ]*\) \([^ ]*\)/\2 \1/' swap.t
xt
student@ai-HP-ProDesk-600-G4-MT:~$ 
```

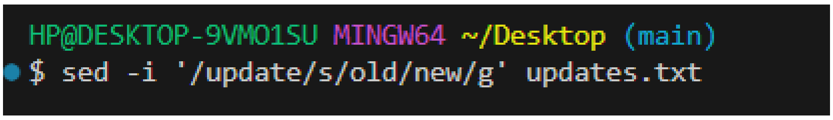
: `^` matches the beginning of the line.

: `\([^ ]*\)` captures the first word (anything that's not a space). :

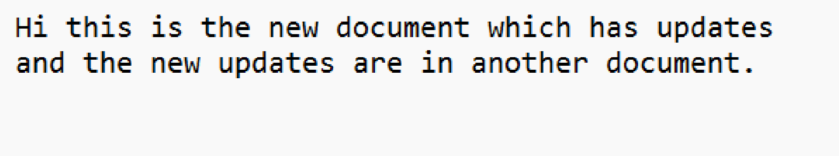
The space between `\([^ ]*\)` and `\([^ ]*\)` separates the first and second words.

: `\2 \1` swaps the first and second words (where `\1` is the first word and `\2` is the second).Remove everything after the first comma in each line of `csv_data.txt`.

7. Replace the word **"old"** with **"new"**, but only on lines that contain the word **"update"**.

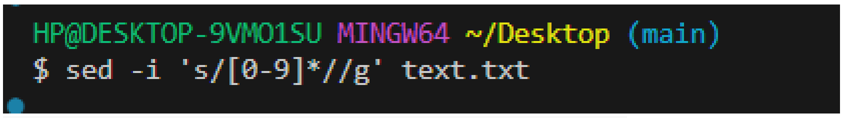


```
HP@DESKTOP-9VMO1SU MINGW64 ~/Desktop (main)
$ sed -i '/update/s/old/new/g' updates.txt
```

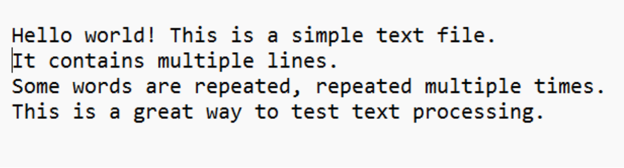


```
Hi this is the new document which has updates
and the new updates are in another document.
```

8. Delete all occurrences of a number from `text.txt`.

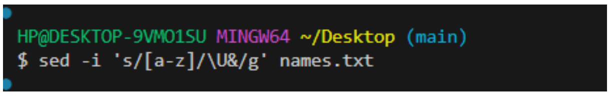


```
HP@DESKTOP-9VMO1SU MINGW64 ~/Desktop (main)
$ sed -i 's/[0-9]*//g' text.txt
```



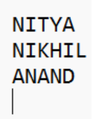
```
Hello world! This is a simple text file.
It contains multiple lines.
Some words are repeated, repeated multiple times.
This is a great way to test text processing.
```

9. Convert all lowercase letters to uppercase in `names.txt`.



```
HP@DESKTOP-9VMO1SU MINGW64 ~/Desktop (main)
$ sed -i 's/[a-z]/\U&/g' names.txt
```

LAB TASK:423132-11/02/2025

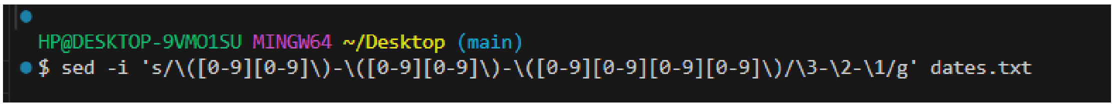


```
NITYA
NIKHIL
ANAND
```

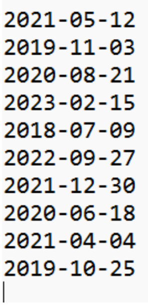
- `\U` tells `sed` to convert the matched character to uppercase.
- `&` represents the matched character (in this case, the lowercase letter).
- `g` at the end makes the substitution **global**, meaning it will replace all occurrences in the file, not just the first one.

`-i`: This flag tells `sed` to edit the file **in place**, meaning it will modify the `names.txt` file directly.

10. Replace all **dates in DD-MM-YYYY format** with **YYYY-MMDD** in `dates.txt`.

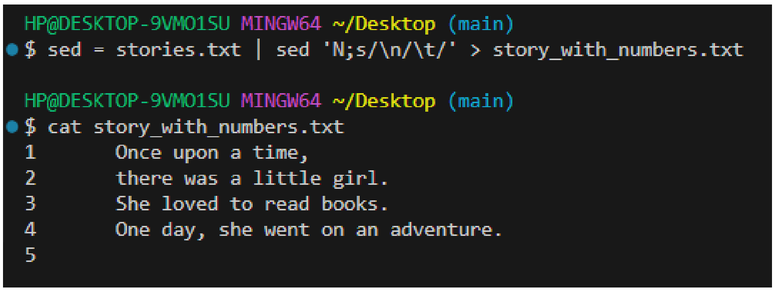


```
HP@DESKTOP-9VMO1SU MINGW64 ~/Desktop (main)
$ sed -i 's/\([0-9][0-9]\)-\([0-9][0-9]\)-\([0-9][0-9][0-9][0-9]\)/\3-\2-\1/g' dates.txt
```



```
2021-05-12
2019-11-03
2020-08-21
2023-02-15
2018-07-09
2022-09-27
2021-12-30
2020-06-18
2021-04-04
2019-10-25
```

11. Add **line numbers** at the beginning of each line in `story.txt`.



```
HP@DESKTOP-9VMO1SU MINGW64 ~/Desktop (main)
$ sed = stories.txt | sed 'N;s/\n/\t/' > story_with_numbers.txt

HP@DESKTOP-9VMO1SU MINGW64 ~/Desktop (main)
$ cat story_with_numbers.txt
1       Once upon a time,
2       there was a little girl.
3       She loved to read books.
4       One day, she went on an adventure.
5
```

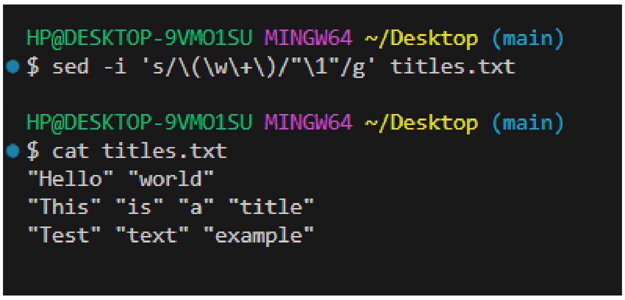
`N`: Reads the next line into the pattern space.

Redirects the output to a new file called `story_with_numbers.txt`. `s/\n/\t/`: Replaces the newline (`\n`) with a tab (`\t`), ensuring the line number and content are on the same line.

12. Surround all words in `title.txt` with double quotes (").

LAB TASK:423132-11/02/2025

- `\(\w\+\)`: This matches a **word** (a sequence of alphanumeric characters). The parentheses `\(...\)` capture the word as a group (which is referred to as `\1`).
- `"\1"`: This replaces the matched word with the word surrounded by double quotes.
- `g`: This flag makes the substitution **global**, meaning it will apply to all words in the file.



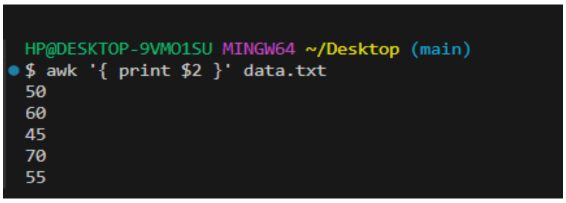
```
HP@DESKTOP-9VMO1SU MINGW64 ~/Desktop (main)
$ sed -i 's/\(\w\+\)/"\1"/g' titles.txt

HP@DESKTOP-9VMO1SU MINGW64 ~/Desktop (main)
$ cat titles.txt
"Hello" "world"
"This" "is" "a" "title"
"Test" "text" "example"
```

Using `awk`– used for pattern scanning, text manipulation, and processing data

1. Print only the second column from a space-separated file (`data.txt`).

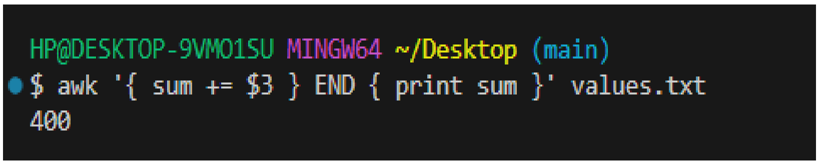
   `{ print $2 }`: This tells `awk` to print the second column of each line.



```
HP@DESKTOP-9VMO1SU MINGW64 ~/Desktop (main)
$ awk '{ print $2 }' data.txt
50
60
45
70
55
```

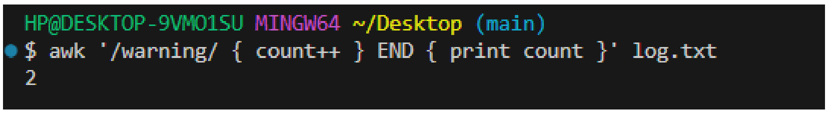
2. Sum the numbers in the third column of `values.txt`.

   `sum += $3`: This adds the value in the third column to the variable `sum`



```
HP@DESKTOP-9VMO1SU MINGW64 ~/Desktop (main)
$ awk '{ sum += $3 } END { print sum }' values.txt
400
```

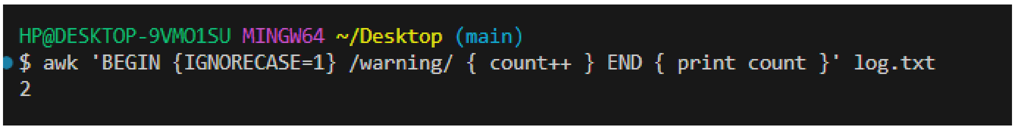
3. Count the number of lines in `log.txt` that contain the word **"warning"**.



```
HP@DESKTOP-9VMO1SU MINGW64 ~/Desktop (main)
$ awk '/warning/ { count++ } END { print count }' log.txt
2
```

For case insensitivity: `IGNORECASE` variable- will treat all pattern matches as case-insensitive

LAB TASK:423132-11/02/2025

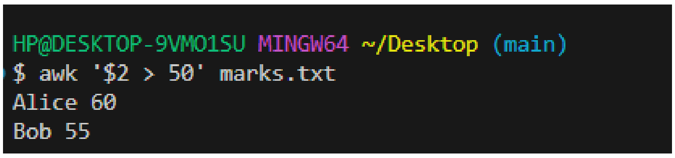


```
HP@DESKTOP-9VMO1SU MINGW64 ~/Desktop (main)
$ awk 'BEGIN {IGNORECASE=1} /warning/ { count++ } END { print count }' log.txt
2
```

4. Print all lines in `marks.txt` where the second column is greater than 50.

**You don't need to write `{ print }` unless you want to perform additional actions

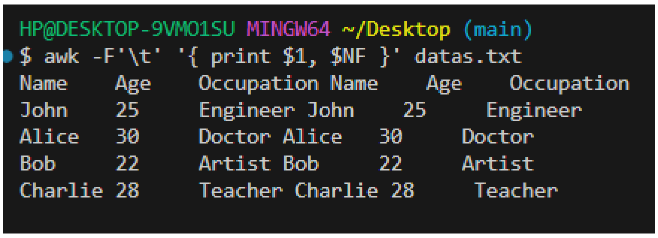
If the condition is **true**, `awk` automatically prints the **entire line** (because the action `{ print }` is implied by default).



```
HP@DESKTOP-9VMO1SU MINGW64 ~/Desktop (main)
$ awk '$2 > 50' marks.txt
Alice 60
Bob 55
```

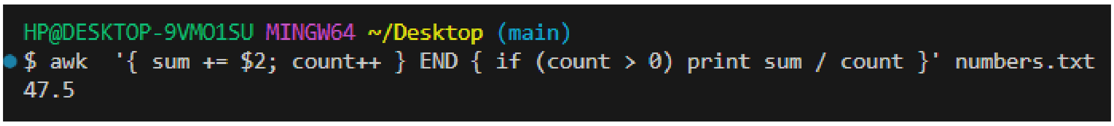
5. Print only the first and last columns from a tab-separated file (`data.csv`).

`-F'\t'`: This option tells `awk` to use a **tab character as the field separator. This is necessary because the file is tab-separated (not space-separated). `'\t'` is the escape sequence for a tab character.



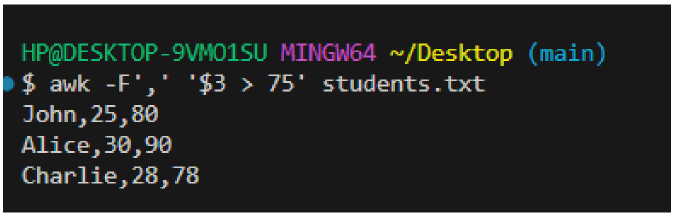
```
HP@DESKTOP-9VMO1SU MINGW64 ~/Desktop (main)
$ awk -F'\t' '{ print $1, $NF }' datas.txt
Name     Age     Occupation Name     Age     Occupation
John     25      Engineer John     25      Engineer
Alice    30      Doctor Alice    30      Doctor
Bob      22      Artist Bob      22      Artist
Charlie 28      Teacher Charlie 28      Teacher
```

6. Calculate and print the average of the numbers in the second column of `numbers.csv`.



```
HP@DESKTOP-9VMO1SU MINGW64 ~/Desktop (main)
$ awk  '{ sum += $2; count++ } END { if (count > 0) print sum / count }' numbers.txt
47.5
```

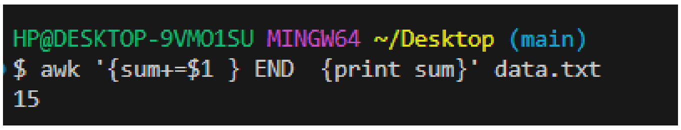
7. Print all lines in `students.csv` where the **third column (marks)** is greater than **75**.



```
HP@DESKTOP-9VMO1SU MINGW64 ~/Desktop (main)
$ awk -F',' '$3 > 75' students.txt
John,25,80
Alice,30,90
Charlie,28,78
```

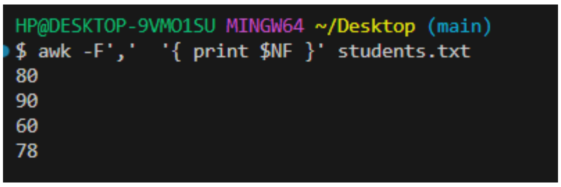
LAB TASK:423132-11/02/2025

8. Print the sum of all numbers in the first column of `data.txt`.



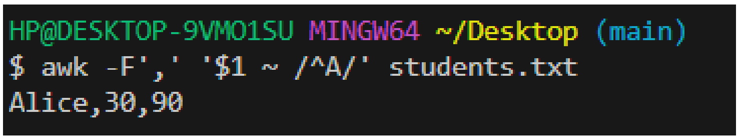
```
HP@DESKTOP-9VMO1SU MINGW64 ~/Desktop (main)
$ awk '{sum+=$1 } END  {print sum}' data.txt
15
```

9. Display the last column of `students.csv`, where columns are separated by commas.



```
HP@DESKTOP-9VMO1SU MINGW64 ~/Desktop (main)
$ awk -F','  '{ print $NF }' students.txt
80
90
60
78
```

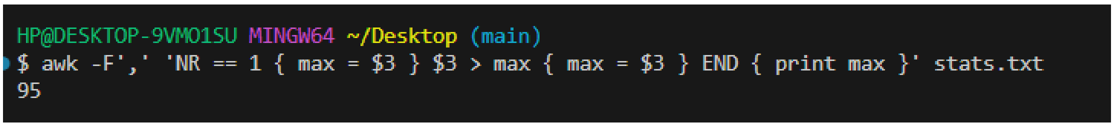
10. Print lines where the second column starts with the letter "A".



```
HP@DESKTOP-9VMO1SU MINGW64 ~/Desktop (main)
$ awk -F',' '$1 ~ /^A/' students.txt
Alice,30,90
```

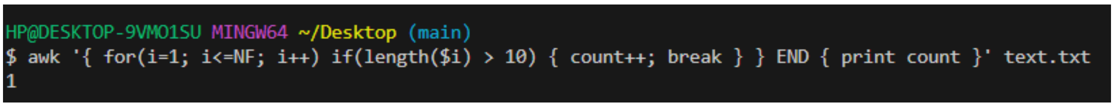
11. Find the **highest number** in the third column of `stats.txt`.

`NR == 1 { max = $3 }`: On the first line (i.e., `NR == 1`), initialize the variable `max` with the value of the third column (`$3`).



```
HP@DESKTOP-9VMO1SU MINGW64 ~/Desktop (main)
$ awk -F',' 'NR == 1 { max = $3 } $3 > max { max = $3 } END { print max }' stats.txt
95
```

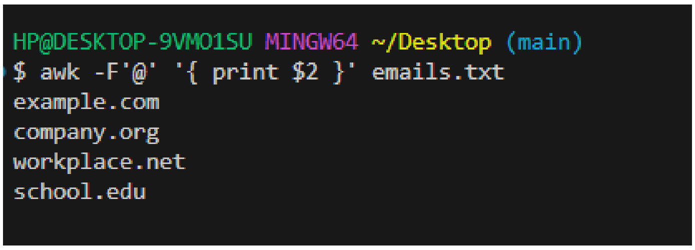
12. Count how many lines contain a word longer than 10 characters in `words.txt`.



```
HP@DESKTOP-9VMO1SU MINGW64 ~/Desktop (main)
$ awk '{ for(i=1; i<=NF; i++) if(length($i) > 10) { count++; break } } END { print count }' text.txt
1
```

13. Extract domain names from an email list (`emails.txt`).

LAB TASK:423132-11/02/2025



```
HP@DESKTOP-9VMO1SU MINGW64 ~/Desktop (main)
$ awk -F'@' '{ print $2 }' emails.txt
example.com
company.org
workplace.net
school.edu
```

Additional exercises

1. Extract all IP addresses from `server.log` using `grep` and format them using `sed`.

   grep -oE '\b([0-9]{1,3}\.){3}[0-9]{1,3}\b' server.log | sed 's/^/IP: /'

2. Find the most frequently occurring word in `words.txt` using `awk`.

awk '{for(i=1;i<=NF;i++) word[$i]++} END {for (w in word) if(word[w]>max) {max=word[w]; most_frequent=w} print most_frequent, max}' words.txt

3. Extract lines from `log.txt` that contain "ERROR" and replace "ERROR" with "ALERT" using `sed`, then save to `alerts.txt`.

sed '/ERROR/s/ERROR/ALERT/g' log.txt > alerts.txt

4. Extract only **IP addresses** from `server.log`, sort them, and remove duplicates.

grep -oP '\b(?:\d{1,3}\.){3}\d{1,3}\b' server.log | sort | uniq

5. Find all words in `document.txt` that appear **more than once** (word frequency count).

tr -cs '[:alnum:]' '[\n*]' < document.txt | sort | uniq -c | awk '$1 > 1'

6. Replace **tab characters with commas** in a tab-separated file (`data.tsv`).

sed 's/\t/,/g' data.tsv > data.csv

7. Print the **top 5 most occurring words** in `essay.txt`.

tr -cs '[:alnum:]' '[\n*]' < essay.txt | sort | uniq -c | sort -nr | head -n 5

LAB TASK:423132-11/02/2025

8. Extract only lines **5 to 15** from a large file (`bigdata.txt`) using `sed` or `awk`.

```
awk 'NR>=5 && NR<=15' bigdata.txt
```

LAB TASK:423132-11/02/2025

9. Count the number of times each unique word appears in `book.txt` (Use `awk` and `sort`).

```
tr -cs '[:alnum:]' '[\n*]' < book.txt | sort | uniq -c | sort –nr
```

10. Find all **unique email domains** in `emails.txt` (e.g., `@gmail.com`, `@yahoo.com`).

```
awk -F '@' '{print "@" $2}' emails.txt | sort | uniq
```