Regular Expressions, Sed, and Awk commands
Regular Expressions

- Sequence of characters that form a search pattern for strings.
- Used in unix commands (grep, sed, awk, find), and high level computer languages (Java, Python, JavaScript, C, C++)
Metacharacters

- Have a special meaning other than the normal meaning.

- Examples (*, ^, +, ., $, ?, {, }, (, ), [ , ], \ )

- Use a preceding backslash “\” to drop the special meaning.
List of Meta Characters

- **Dot ( . )**
  Matches any character except a newline “\n”.

- **Caret ( ^ )**
  Matches the beginning of input string.

- **Dollar sign ( $ )**
  Matches the ending of input string or just before the newline character.

- **Asterisk ( * )**
  Matches the preceding subexpression zero or more times.
List of Meta Characters (cont..)

- **Plus ( + )**
  Matches the preceding subexpression one or more times.

- **Question mark ( ? )**
  Matches the preceding subexpression zero or one time.
Character Sets

- [a-z]
  Matches any simple letter.

- [A-Z]
  Matches any capital letter.

- [0-9]
  Matches any number.

- "[“ and “]” can be used to indicate any range of characters. ( [5-9] will match digits from 5 to 9)
Repetitions

- \{n\}
  Matches preceding subexpression exactly n times.

- \{n,\}
  Matches preceding subexpression at least n times.

- \{n,m\}
  Matches at least n and at most m times.

- n and m should be positive integer values
Examples

- \{n\}
  Matches preceding subexpression exactly n times.
- \{n,\}
  Matches preceding subexpression at least n times.
- \{n,m\}
  Matches at least n and at most m times.
- n and m should be positive integer values
Sed (Stream Editor)

- One of the early Unix commands built for command line processing of data files.
- Reads text, line by line, from an input stream or file, then execute command in the current line.
- Can be used to search, substitute, delete, append files.
- Useful tool for manipulating large data files in unix environments.
Search

- Searches for the line with matching text. The text need to be searched for is included within slashes “/”.
- Syntax :

```bash
sed -n '/text_to_search/p' input_file >> output_file
```

- “-n” stands to suppress printing all the input lines.
- “p” stand for “print”.
Search With Regular Expression

- The regular expression need to be searched out is included within slashes “/”
- Syntax:
  ```bash
  sed -n -r '/regex/p' input_file >> output_file
  ```
- “-r” stands for regular expression.
Similar to searching for a text.

Syntax:

```
sed '/text_to_search/d' input_file >> output_file
```

- "d" stand for "delete".
- "-n" is dropped in this syntax.
- Can search regular expressions and delete matching lines
Append and Insert

- Add a new line after (append) or before (insert) the matching line.
- Syntax:

  ```bash
  sed '/text_to_search/a line_append' input_file >> output_file
  ```

- Syntax:

  ```bash
  sed '/text_to_search/i line_insert' input_file >> output_file
  ```
AWK Command

- AWK is an interpreted text processing and reporting language.
- Useful tool for data files with rows and columns.
- Process line by line from the input stream.
- AWK was created at Bell Labs in the 1970s,
Basic Syntax

- The Basic syntax contains the pattern to match in each line and the action for the matching lines.

```
awk [options] 'pattern {action}' input_file >> output_file
```

- The Basic syntax contains the pattern to match in each line and the action for the matching lines.
Useful In-build Variables

- NR : Row number
- NF : Column (field) number
- RS : Row separator
- FS : Column (field) separator
- FILENAME : input file name
Search Lines

- Searches for the line with matching text. The text need to be searched for is included within slashes “/”.

- Syntax:
  
  \texttt{awk `/text\_to\_search/ \{print\}` input\_file >> output\_file}

- Regular expression can be used for searching.
Print Rows within Matching Lines

- Output all the lines between the first matching of the first text and first matching of the second text.

  ```bash
  awk '/starting_text/, /ending_text/ {print}' input_file >> output_file
  
  awk 'NR==10, NR==20 {print}' input_file >> output_file
  ```

- Line numbers can be used for marking the beginning and end of the lines need to be output.
Select Columns for Output

- Different columns can be selected for the output file.
- Syntax:
  ```
  awk '{print $2, $4, $6, $NF}' input_file >> output_file
  ```
Filtering Data

● Data lines can be filtered depending on the values of data fields.
● Syntax:

```
awk -F, \$2 > 80 \{print\} \ input_file >> output_file
```

● Using if statement.

```
awk -F'\t' \{if($2>80 && $2<95) print;\} \ input_file >> output_file
```

● “-F” denote the field separator.
Filtering Data (Cont..)

- Data lines can be filtered depending on the text in a field.
- Syntax:

  ```awk -F, '$2 ~/text_to_match/’ input_file >> output_file```

- The “print” action can be dropped since “print” is the default action.
BEGIN and END Keywords

- **Syntax:**
  ```bash
  awk 'BEGIN {beginning action}
  { pattern action;}
  END {ending actions}'
  input_file >> output_file
  ```

- Beginning actions are executed before the first line is read.
- Ending actions are executed after the last line is read.
Use counters

- Can be used to count the number of data entries with a given property.

- Syntax:

```bash
awk -F, 'BEGIN {count=0;}
        $4 >80 {count++;}
        END {print "Number of lines = ", count;}
' input_file
```
Find Average of a Field

- Can be used to find the average of values in a column.

- Syntax:

```bash
awk -F, 'BEGIN {sum=0; count=0;} 
{sum+=$4; count++;} 
END {print "Average = ", sum/count;}'
input_file
```
Find Average of a Field (cont..)

- Can be used to find the average of values in a column only for the data entries satisfying a given condition.

- Syntax:

```bash
awk -F, 'BEGIN {sum=0; count=0;} if($1 >1950) {sum+=$4; count++;} END {print "Average = ", sum/count;}' input_file
```
THANK YOU !!!