8.3 File Analysis

8.3.1 Regular Expressions

Regular Expressions

A **regular expression (regex)** is a sequence of characters that define a pattern, which can be used for searching text. The most basic regular expression is a word. To illustrate the concept of regular expressions, patterns and matches of regular expressions to the following text will be shown:

Apples are a fruit. An apple a day keeps the doctor away.

To search this piece of text for occurrences of the word "apple" in lower case, the regular expression would simply be "apple". The matches of the regular expression are shown in the text below:

Apples are a fruit. An apple a day keeps the doctor away.

A more complex regular expression could search for the word "apple" and match all occurrences that start with an upper or lower case "A". The regular expression for this task is "[Aa]pple". The matches are shown in the text below:

Apples are a fruit. An apple a day keeps the doctor away.

As you can see, the whole word does not need to match the regex in order for there to be a match. A regular expression can be written to match any 5-letter character sequence containing no spaces, numbers, or symbols. The regular expression for this task is "[A-Za-z]{5}". The matches are shown in the text below:

Apples are a fruit. An apple a day keeps the doctor away.

The rules of writing a regular expression are complex. In this course, regular expressions will not be covered in depth, further information can be found online

The two basic aspects of writing a regular expression are defining what to match and how many times to match it. The following tables describe some of the basic symbols for writing regular expressions:

Representations of Multiple Characters

`0-9`	Any Number
`A-Z`	Any uppercase letters
`a-z`	Any lowercase letters
`.`	Any character

Quantification of Occurrences

`*`	Zero or more occurrences
`+`	One or more occurrences
`?`	Zero or one occurrence
`{N}`	N occurrences

In regular expressions, square brackets `[]` are used to denote a set of characters to match.

To understand how to put these symbols into a regular expression, below are descriptions of patterns that can be used to find matches, along with the regular expression.

Example 1: match the number 8004

Pattern: one occurrence of the number 8, followed by two occurrences of the number 0, followed by one occurrence of the number 4.

Regular expression: `8004`

Example 2: Match any number from 8000-8999

Pattern: one occurrence of the number 8, followed by 3 occurrences of any number from 0 to 9.

Regular expression: `8[0-9]{3}`

Example 3: Match any word that starts with a capital letter

Pattern: one occurrence of any capital letter from "A" to "Z", followed by zero or more occurrences of any lowercase letter from "a" to "z".

Regular expression: `[A-Z][a-z]*`

In example 1, the pattern needs to describe an exact match to the number 8004, thus the regular expression is the exact number. Any time an exact set of characters is being matched the regular expression is simply the set of characters.

In example 2, `[0-9]` represents the set of numbers. This is directly followed by `{3}`, indicating that three occurrences of characters that belong to the set `[0-9]` need to be present for a match. The symbols used to define the quantification of occurrences only apply to the character or set of characters directly to the left of the quantification symbol. For example, the regular expression `ab+` matches one "a" followed by one or more "b"s (`+` indicates 1 or more occurrences). The expression `ab+` will match `ab` or `abbbb` but will not match `aaab`.

In example 3, `[A-Z]` represents the set of capital letters and `[a-z]` represents the set of lowercase letters. `[A-Z]` is not followed by any symbols defining the expected number of occurrences meaning that exactly one occurrence is expected. `[a-z]` is followed by an asterisk (`*`) indicating that zero or more occurrences of lowercase letters can be present. The asterisk is not applied to the set of uppercase letters as it is only applied to the set directly to the left.

Below are three more examples of regular expressions along with a subset of possible matches and non-matches, these examples are not exhaustive, there are hundreds to thousands of possible matches for each regular expression and even more non-matches.

Regular expression: `[xyz]+`

The regular expression will match one or more occurrence of the letters "x", "y", and "z". By putting the letters within square brackets, it becomes a set. Therefore, one or more occurrences of any combination of the letters within the set will match.

Matches: `xxxx`, `yyy`, `zzzzz`, `zzyx`, `xyz`, `x`, `y`, `z`

Non-matches: `aazzyx`, `abc`, `ccyc`, `dd`

Regular expression: `pt_[A-Za-z0-9]*`

The regular expression will match "pt_" followed by 0 or more occurrences of uppercase letters, lowercase letters, and numbers. Multiple ranges can put into one set.

Matches: `pt_8xdC7`, `pt_`, `pt_100`, `pt_A`, `pt_onehundred`
Non-matches: `pt89cD`, `t_8xdC7`, `pt_one_hundred`, `100`

Regular expression: `[0-9]{2}_[A-Z]?`

The regular expression will match two numbers, followed by an underscore "_", followed by 0 or 1 occurrence of an uppercase letter.
Matches: `13_C`, `78_`, `94_X`, `00_`
Non-matches: `3_C`, `13C`, `94_XB`, `C_13`

The grep Command

Regular expressions can be used with the `grep` command.

```
grep [OPTION]... PATTERN [FILE]...
```

The `grep` command searches for matches to a specified pattern in a file. The `PATTERN` argument is a regular expression.

The directory Week.8/8.3.File.Analysis contains a file called test_grep.txt with 5 lines, each containing 4 lowercase letters:

```
j:~$ cd Week.8/8.3.File.Analysis
j:~/Week.8/8.3.File.Analysis$ cat test_grep.txt
aaaa
aazz
yyxz
xxxx
abcd
j:~/Week.8/8.3.File.Analysis$
```

By default, the `grep` command will return all lines that contain a match to the `PATTERN` anywhere in the line. For example, if the pattern is `"a"`, all lines containing a single occurrence of the letter "a" will be output.

```
j:~/Week.8/8.3.File.Analysis$ grep "a" test_grep.txt
aaaa
aazz
abcd
j:~/Week.8/8.3.File.Analysis$
```

Comparing back to the full contents of the file above, the 1st, 2nd, and 5th lines contain at least one lowercase "a" and were output.

The `grep` command has many useful options. One important option is the `-P` (`--perl-regexp`) option which allows one to use more complex regular expressions. Some special regular expression characters are not recognized by standard `grep`, but are recognized by Perl regular expressions, including: `?`, `+`, `{}`, `\t`.

```
  -P, --perl-regexp        PATTERNS are Perl regular expression
```

To search for matches to the regular expression "[xyz]+", the -P option is required. As a reminder, this regular expression will match one or more occurrence of the letters "x", "y", and "z".

```
j:~/Week.8/8.3.File.Analysis$ grep -P "[xyz]+" test_grep.txt
aazz
yyxz
xxxx
j:~/Week.8/8.3.File.Analysis$
```

Notice that because grep searches for matches to the pattern within each line, the line aazz is output because zz matches the pattern.

The -o option returns only the part of the line that matches the pattern.

```
  -o, --only-matching      show only nonempty parts of lines that match
```

In 8.2.1 using multiple options is introduced. For a command that has the options -a, -b, and -c, if none of the options take arguments, there are two ways to use the options together:

```
command -a -b -c ARGUMENTS

command -abc ARGUMENTS
```

When multiple options without arguments are used they can be combined without spaces after a single hyphen.

To output only the parts of test_grep.txt that match the pattern "[xyz]+", both the -P and the -o option are required:

```
j:~/Week.8/8.3.File.Analysis$ grep -oP "[xyz]+" test_grep.txt
zz
yyxz
xxxx
j:~/Week.8/8.3.File.Analysis$
```

Using the -o option, only the zz from the line containing aazz is output.

The -x option only returns a line with a match if the whole line matches the pattern.

```
  -x, --line-regexp          match only whole lines
```

Using the -x option, the line containing aazz is not returned at all as only part of the line matches the pattern.

```
j:~/Week.8/8.3.File.Analysis$ grep -xP "[xyz]+" test_grep.txt
yyxz
xxxx
j:~/Week.8/8.3.File.Analysis$
```

There are many other options to use with `grep`, including:

```
-n, --line-number        print line number with output lines
-v, --invert-match       select non-matching lines
```

Output lines that match the pattern "`[xyz]+`" along with their line numbers:

```
j:~/Week.8/8.3.File.Analysis$ grep -nP "[xyz]+" test_grep.txt
2:aazz
3:yyxz
4:xxxx
j:~/Week.8/8.3.File.Analysis$
```

Output lines that DO NOT match the pattern "`[xyz]+`" along with their line numbers:

```
j:~/Week.8/8.3.File.Analysis$ grep -nvP "[xyz]+" test_grep.txt
1:aaaa
5:abcd
j:~/Week.8/8.3.File.Analysis$
```

Combining the `-n` and `-o` options will return all the unique occurrences of a pattern along with their line numbers. For example, using the pattern "`a`":

```
j:~/Week.8/8.3.File.Analysis$ grep -no "a" test_grep.txt
1:a
1:a
1:a
1:a
2:a
2:a
5:a
j:~/Week.8/8.3.File.Analysis$
```

There are four matches to "`a`" on line 1, two on line 2, and one on line 5.

Patterns in Biological Data

The `grep` command is very useful for gathering information from biological data files in the command line. For example, grep can be used to learn about the tab delimited file `cancer_pts.txt`. The file has 4 columns: patient.ID, sex, age, and cancer.type.

```
j:~/Week.8/8.3.File.Analysis$ head cancer_pts.txt
patient.ID      sex     age     cancer.type
DO49528 Male    54      Acute myeloid leukemia
DO49529 Male    54      Acute myeloid leukemia
DO49531 Male    49      Acute myeloid leukemia
DO49532 Female  39      Acute myeloid leukemia
DO49533 Male    39      Acute myeloid leukemia
DO49536 Male    46      Acute myeloid leukemia
```

```
DO49537 Female  50      Acute myeloid leukemia
DO52732 Female  75      Acute myeloid leukemia
DO52739 Female  66      Acute myeloid leukemia
j:~/Week.8/8.3.File.Analysis$
```

For example, all the lines in `cancer_pts.txt` that contain the word neoplasm (capitalized or all lowercase) can be found using the regular expression "`[Nn]eoplasm`":

```
j:~/Week.8/8.3.File.Analysis$ grep "[Nn]eoplasm" cancer_pts.txt | head
DO49137 Female  67      Intraductal papillary mucinous neoplasm
DO49172 Female  60      Intraductal papillary mucinous neoplasm
DO33488 Male    61      Intraductal papillary mucinous neoplasm
DO6428  Female  78      Mature B-Cell Neoplasms
DO7018  Male    60      Mature B-Cell Neoplasms
DO27847 Male    16      Mature B-Cell Neoplasms
DO51967 Female  54      Mature B-Cell Neoplasms
DO51989 Female  61      Mature B-Cell Neoplasms
DO51992 Female  42      Mature B-Cell Neoplasms
DO52651 Female  72      Mature B-Cell Neoplasms
j:~/Week.8/8.3.File.Analysis$
```

The output contains patients with the cancer type "Intraductal papillary mucinous neoplasm", as well as patients with the cancer type "Mature B-Cell Neoplasms". The `[Nn]` in the regular expression allows the command to return patients with both types of cancer.

In the output of the `grep` command above, patients IDs vary in length. Each starts with the letters "DO" followed by numbers. The regular expression "`DO[0-9]{3}`" will retrieve all the lines containing DO followed by 3 numbers. Note that use of curly braces requires the `-P` option.

```
j:~/Week.8/8.3.File.Analysis$ grep -P "DO[0-9]{3}" cancer_pts.txt | head
DO49528 Male    54      Acute myeloid leukemia
DO49529 Male    54      Acute myeloid leukemia
DO49531 Male    49      Acute myeloid leukemia
DO49532 Female  39      Acute myeloid leukemia
DO49533 Male    39      Acute myeloid leukemia
DO49536 Male    46      Acute myeloid leukemia
DO49537 Female  50      Acute myeloid leukemia
DO52732 Female  75      Acute myeloid leukemia
DO52739 Female  66      Acute myeloid leukemia
DO52740 Male    45      Acute myeloid leukemia
j:~/Week.8/8.3.File.Analysis$
```

The patient IDs in the output do not contain only three numbers. This is because grep with the pattern "`DO[0-9]{3}`" will return lines that contain the pattern anywhere in the line, even if it is followed directly by more numbers. To get the lines that have patient IDs with ONLY three numbers there are two possibilities.

Firstly, because this is a tab-delimited file, the patient ID will always be followed by a tab. Extending the regular expression "`DO[0-9]{3}\t`" will find "DO", followed by three letters, followed directly by a tab:

```
j:~/Week.8/8.3.File.Analysis$ grep -P "DO[0-9]{3}\t" cancer_pts.txt |
head
DO472   Male    53      Bladder Cancer
DO477   Male    79      Bladder Cancer
DO479   Male    34      Bladder Cancer
DO483   Female  67      Bladder Cancer
DO496   Female  53      Bladder Cancer
DO498   Male    84      Bladder Cancer
DO522   Female  64      Bladder Cancer
DO536   Male    65      Bladder Cancer
DO548   Male    57      Bladder Cancer
DO555   Female  79      Bladder Cancer
j:~/Week.8/8.3.File.Analysis$
```

Secondly, one can use the same pattern as was used originally ("`DO[0-9]{3}`") along with the `grep` option `-w`:

```
  -w, --word-regexp       match only whole words
```

This option only outputs matches that are preceded by a space, a tab, or the beginning of a line and followed by a space, a tab, or the end of a line.

```
j:~/Week.8/8.3.File.Analysis$ grep -wP "DO[0-9]{3}" cancer_pts.txt |
head
DO472   Male    53      Bladder Cancer
DO477   Male    79      Bladder Cancer
DO479   Male    34      Bladder Cancer
DO483   Female  67      Bladder Cancer
DO496   Female  53      Bladder Cancer
DO498   Male    84      Bladder Cancer
DO522   Female  64      Bladder Cancer
DO536   Male    65      Bladder Cancer
DO548   Male    57      Bladder Cancer
DO555   Female  79      Bladder Cancer
j:~/Week.8/8.3.File.Analysis$
```

8.3.2 Counting Occurrences

Line Counts

The `wc` command counts the number of lines, words, and/or bytes in a file.

```
wc [OPTION]... [FILE]...
```

Used without options, it returns the counts for lines, words, and bytes, in that order:

```
j:~/Week.8/8.3.File.Analysis$ wc cancer_pts.txt
 2344 12318 86042 cancer_pts.txt
j:~/Week.8/8.3.File.Analysis$
```

The file `cancer_pts.txt` has 2,344 lines, 12,318 words, and 86042 characters. Remember that the number of bytes is the number of characters including line ends, spaces, and tabs.

The `wc` command can also be used with one of the following options to only output the number of bytes, words, or lines:

```
  -c, --bytes            print the byte counts
  -m, --chars            print the character counts
  -l, --lines            print the newline counts
```

```
j:~/Week.8/8.3.File.Analysis$ wc -l cancer_pts.txt
2344 cancer_pts.txt
j:~/Week.8/8.3.File.Analysis$
```

Used with the wildcard, the number of lines in all files in the PWD that end with `.txt` can be output, as well as the total number of lines in all the `.txt` files:

```
j:~/Week.8/8.3.File.Analysis$ wc -l *.txt
 2344 cancer_pts.txt
    5 test_grep.txt
 2349 total
j:~/Week.8/8.3.File.Analysis$
```

By piping the output of a `grep` command to the `wc` command, the number of lines containing a specific pattern in a file can be identified. For example, count the number of lines in `cancer_pts.txt` that contain the word “Glioma”:

```
j:~/Week.8/8.3.File.Analysis$ grep "Glioma" cancer_pts.txt | wc -l
48
j:~/Week.8/8.3.File.Analysis$
```

By piping the output of a `grep` command to the `wc` command, the number of lines containing a specific pattern in a file can be identified. For example, count the number of lines in `cancer_pts.txt` that contain the word “Glioma”:

```
j:~/Week.8/8.3.File.Analysis$ grep "Glioma" cancer_pts.txt | wc -l
48
j:~/Week.8/8.3.File.Analysis$
```

Sorting Files

The `sort` command will sort a file based on a full line or one more fields (columns).

```
sort [OPTION]... [FILE]...
```

By default, the `sort` command sorts a file based on the full line in increasing alphabetical order. For example, sort the `cancer_pts.txt` file.

```
j:~/Week.8/8.3.File.Analysis$ sort cancer_pts.txt | head -n 5
DO1000  Female  61      Breast Cancer
DO1001  Female  41      Breast Cancer
DO1002  Female  39      Breast Cancer
DO1003  Female  34      Breast Cancer
DO1004  Female  59      Breast Cancer
j:~/Week.8/8.3.File.Analysis$
```

The `-r` option tells the `sort` command to sort in reverse (decreasing) order:

```
  -r, --reverse              reverse the result of comparisons
```

```
j:~/Week.8/8.3.File.Analysis$ sort -r cancer_pts.txt | head -n 5
patient.ID      sex     age     cancer.type
DO9940  Female  78      Colorectal Cancer
DO9876  Male    75      Colorectal Cancer
DO9788  Female  73      Colorectal Cancer
DO9732  Female  78      Colorectal Cancer
j:~/Week.8/8.3.File.Analysis$
```

The `-k` option tells the `sort` command to sort based on a "key". This is the number of the field to sort on.

```
  -k, --key=KEYDEF      sort via a key; KEYDEF gives location and type
```

For example, to sort the `cancer_pts.txt` file based on the patient cancer type, the `-k` option is used with the argument 4, because cancer type is in the 4th column/field.

```
j:~/Week.8/8.3.File.Analysis$ sort -k 4 cancer_pts.txt | head -n 5
DO221548        Female  35      Acute myeloid leukemia
DO49528 Male    54      Acute myeloid leukemia
DO49529 Male    54      Acute myeloid leukemia
DO49531 Male    49      Acute myeloid leukemia
DO49532 Female  39      Acute myeloid leukemia
j:~/Week.8/8.3.File.Analysis$
```

By default, `sort` uses alphabetical order which can cause problems when sorting with numbers. Although 9 is a smaller number than 10, in alphabetical order the number 10 comes before the number 9, because it begins with 1. The `cancer_pts.txt` file contains 77 patients under the age of 10, however, when sorted by column 3 the output shows 10-year-old patients first:

```
j:~/Week.8/8.3.File.Analysis$ sort -k 3 cancer_pts.txt | head -n 5
DO35558 Female  10      Embryonal Tumor
DO35584 Male    10      Embryonal Tumor
DO35585 Female  10      Embryonal Tumor
DO35624 Male    10      Embryonal Tumor
DO35737 Male    10      Embryonal Tumor
j:~/Week.8/8.3.File.Analysis$
```

This can be corrected by using the -g option:

```
  -g, --general-numeric-sort  compare according to general numerical
                                value
```

This compares numbers based on numerical value instead of alphabetical value. Sorting the `cancer_pts.txt` file on field 3 again with the `-g` option changes the output.

```
j:~/Week.8/8.3.File.Analysis$ sort -g -k 3 cancer_pts.txt | head -n 5
DO1013  Female  NA      Breast Cancer
DO1014  Female  NA      Breast Cancer
DO1015  Female  NA      Breast Cancer
DO1016  Female  NA      Breast Cancer
DO1076  Female  NA      Breast Cancer
j:~/Week.8/8.3.File.Analysis$
```

A number of patients do not have an associated age and instead have an “NA” value in place of the patient age. Using the `grep` command with the `-v` option we can remove these patients from the output.

```
j:~/Week.8/8.3.File.Analysis$ sort -g -k 3 cancer_pts.txt | grep -v "NA"
| head -n 5
patient.ID      sex     age     cancer.type
DO35566 Male    1       Medulloblastoma
DO35574 Female  1       Medulloblastoma
DO35620 Male    2       Medulloblastoma
DO35622 Male    2       Embryonal Tumor
j:~/Week.8/8.3.File.Analysis$
```

Now patients are in order of age, starting with 1. The patients can also be output in reverse order of age:

```
j:~/Week.8/8.3.File.Analysis$ sort -gr -k 3 cancer_pts.txt | grep -v
"NA" | head -n 5
DO47863 Female  90      Esophagogastric Cancer
DO43378 Female  90      Uterine Endometrioid Carcinoma
DO33200 Female  90      Pancreatic Cancer
DO45185 Male    89      Hepatobiliary Cancer
DO37267 Female  89      Colorectal Cancer
j:~/Week.8/8.3.File.Analysis$
```

Unique Lines

The `uniq` command outputs unique **adjacent** lines.

```
uniq [OPTION]... [FILE]...
```

The `uniq` command outputs a single line for every set of adjacent lines that are exactly the same. If applied to a file containing the following lines:

```
aaaaaa
aaaaaa
bbbbbb
bbbbbb
bbbbbb
```

The `uniq` command would output:

```
aaaaaa
bbbbbb
```

Each unique line in the file is output. However, if the `uniq` command was applied to a file containing the following lines:

```
aaaaaa
bbbbbb
bbbbbb
bbbbbb
aaaaaa
```

The `uniq` command would output:

```
aaaaaa
bbbbbb
aaaaaa
```

This is because the in the second file, the two lines containing `aaaaaa` are not adjacent to one another. For example, these are the first 10 lines of the 2nd column of `cancer_pts.txt`:

```
j:~/Week.8/8.3.File.Analysis$ cut -f 2 cancer_pts.txt | head
sex
Male
Male
Male
Female
Male
Male
Female
Female
Female
j:~/Week.8/8.3.File.Analysis$
```

Only three unique values are present: “sex”, “Male”, and “Female”. Applying the `uniq` command to these 10 lines:

```
j:~/Week.8/8.3.File.Analysis$ cut -f 2 cancer_pts.txt | head | uniq
sex
Male
Female
Male
Female
j:~/Week.8/8.3.File.Analysis$
```

To prevent this issue, the `uniq` command should always be used with the `sort` command:

```
j:~/Week.8/8.3.File.Analysis$ cut -f 2 cancer_pts.txt | head | sort |
uniq
Female
Male
sex
j:~/Week.8/8.3.File.Analysis$
```

When the `sort` command is used all of the identical lines will be adjacent to one another.

The `-c` option tells the `uniq` command to count the number of lines that have each unique value:

```
  -c, --count           prefix lines by the number of occurrences
```

For example, to get the number of patients in `cancer_pts.txt` that have each type of cancer, the 4th column (cancer.type) can be extracted with the `cut` command, then the `sort` and `uniq` commands can be applied:

```
j:~/Week.8/8.3.File.Analysis$ cut -f 4 cancer_pts.txt | sort | uniq -c
     13 Acute myeloid leukemia
     23 Bladder Cancer
     47 Bone Cancer
    209 Breast Cancer
      1 cancer.type
     20 Cervical Cancer
     52 Colorectal Cancer
    120 Embryonal Tumor
     20 Endometrial Cancer
    165 Esophagogastric Cancer
     13 Essential Thrombocythemia
     48 Glioma
     56 Head and Neck Cancer
    348 Hepatobiliary Cancer
      6 Intraductal papillary mucinous neoplasm
     38 Lung Cancer
      1 Mastocytosis
    103 Mature B-cell lymphoma
     93 Mature B-Cell Neoplasms
     21 Medulloblastoma
     11 Myelodysplastic/Myeloproliferative Neoplasms
     46 Non-Small Cell Lung Cancer
     99 Ovarian Cancer
```

```
    305 Pancreatic Cancer
    194 Prostate Cancer
    186 Renal Cell Carcinoma
     34 Soft Tissue Sarcoma
     48 Thyroid Cancer
     24 Uterine Endometrioid Carcinoma
j:~/Week.8/8.3.File.Analysis$
```

Similarly, the number of patients of each sex can counted:

```
j:~/Week.8/8.3.File.Analysis$ cut -f 2 cancer_pts.txt | sort | uniq -c
   1053 Female
   1290 Male
      1 sex
j:~/Week.8/8.3.File.Analysis$
```

Note that in the last two commands, a line with the value of the header file was returned (cancer.type & sex). The `tail` command can be used to get all lines in a file after the first line without having to calculate the number of lines in a file. The following command returns every line in a file except for the first line:

```
tail -n +2 FILE
```

Re-running the above command to get the number of patients of each sex with the `tail` command, the header row can be ignored:

```
j:~/Week.8/8.3.File.Analysis$ tail -n +2 cancer_pts.txt | cut -f 2 |
sort | uniq -c
   1053 Female
   1290 Male
j:~/Week.8/8.3.File.Analysis$
```

Applying Commands to Answer Questions

Below are a number of example commands that answer questions about the information in the `cancer_pts.txt` file by combining the commands from this module and module 8.2.

What is the age of the youngest patient with breast cancer?

```
j:~/Week.8/8.3.File.Analysis$ grep "Breast Cancer" cancer_pts.txt | sort
-g -k 3 | grep -v "NA" | head -n 1
DO1257  Female  30      Breast Cancer
j:~/Week.8/8.3.File.Analysis$
```

What is the age of the oldest patient with ovarian cancer?

```
j:~/Week.8/8.3.File.Analysis$ grep "Ovarian Cancer" cancer_pts.txt |
sort -gr -k 3 | grep -v "NA" | head -n 1
DO28093 Female  81      Ovarian Cancer
j:~/Week.8/8.3.File.Analysis$
```

How many patients are 75 years old?

```
j:~/Week.8/8.3.File.Analysis$ cut -f 3 cancer_pts.txt | grep -w "75" |
wc -l
49
j:~/Week.8/8.3.File.Analysis$
```

What types of cancer do patients under 10 have? (Note: `grep -w "[0-9]"` gets lines that contain a single number in one of the fields, i.e. 1 digit ages.)

```
j:~/Week.8/8.3.File.Analysis$ grep -w "[0-9]" cancer_pts.txt | cut -f 4
| sort | uniq
Embryonal Tumor
Mature B-cell lymphoma
Medulloblastoma
j:~/Week.8/8.3.File.Analysis$
```

How many female and male hepatobiliary cancer patients are there?

```
j:~/Week.8/8.3.File.Analysis$ grep "Hepatobiliary Cancer" cancer_pts.txt
| cut -f 2 | sort | uniq -c
    103 Female
    245 Male
j:~/Week.8/8.3.File.Analysis$
```