General regex and programming

1.) Create regular expressions for lines that match these things

a.) Any number that is a multiple of 5 – [0-9]*[0|5]

b.) Exactly 5 characters - [a-zA-Z]{5}

c.) Any letter followed by a number - [a-zA-Z][0-9]+

d.) The first 6 columns of a BED file – [.+]\t[0-9]+\t[0-9]+\t[.+]\t[1-9][0-9]{1,2}\t[+|-]

chromosome, chomStart, ChromEnd, name, score(between 0 and 1000), strand(+ or -). All tab-delimited.

e.) Write a regular expression that matches the first 3 bases in a DNA sequence - ^[ACGTacgt]{1,3}

f.) Write a regular expression that matches the last 3 bases in a DNA sequence - [ACGTacgt]{1,3}$

g.) Write a regular expression that matches two numbers followed by 2 lower case letters - [0-9]{2}\s[a-z]{2}

h.) What does this regular expression match: `\d*\.\d{3}` – floating point number with precision up to 3 decimal digits

2.) What is the value of `a_number` after each line? (Write the value of a_number after each statement. These statements are in continuation)
a_number = 6
a_number -= 12 : -6
a_number += 15 : 9
a_number /= 3 : 3
a_number *= 2 : 6
a_number = a_number + a_number / 2 : 9

3.) What do these Boolean statements evaluate to? (Answer as simple as TRUE or FALSE)

a.) 1 > 2 -false
b.) 2 > 1 - true
c.) !(2 > 1) - false
d.)(2 > 1) || (1 > 2) - true
e.)(2 > 1) && (1 > 2) - false
f.) !(2 > 1) || !(1 > 2) - true
g.)!((2 > 1) || (1 > 2)) - false

h.)!((2 > 1) && (1 > 2)) - true

Regular Expression Exercises

6.) Searching a file with `grep`

a.) Extract the knownGene.txt.gz. Google the command.

Extracted with the following command :
tannishtha@tsom3:~$ gunzip knownGene.txt.gz

b.) Use `grep` to get all genes on `chr22`

```
tannishtha@tsom3:~$ grep "chr22" knownGene.txt > chr22.txt
```

c)Use `grep` to get all and only those genes that occur on `chr1`

```
tannishtha@tsom3:~$ grep -w "chr1" knownGene.txt > chr1.txt
Here -w will match the lines containing just the word “chr1”.
```

7.) Editing data streams with `sed`

a.) Take the result of 6b and duplicate each line
tannishtha@tsom3:~$ sed 's/.*/&\n&/g' chr1.txt > chr1copy.txt
& returns the matched text.

b.) Change the ‘`chr`’ portion of every other line to ‘`cow`’

```
tannishtha@tsom3:~$ sed '2~2 s/chr/cow/g' chr1copy.txt > cow.txt

This will change the chr portion of every even numbered to cow.
```

c)Delete the lines that have cow in them.
tannishtha@tsom3:~$ sed '/cow/d' cow.txt > cowdel.txt
'd' is used to delete the pattern

d)Do a-c again, but this time do it ‘in-place’ in the file.
-i used to change in-place.
tannishtha@tsom3:~$ sed -i 's/.*/&\n&/g' chr1.txt
tannishtha@tsom3:~$ sed -i '2~2 s/chr/cow/g' chr1.txt
tannishtha@tsom3:~$ sed -i '/cow/d' chr1.txt

Real meat

4.) Format recognition

A lot of bioinformatics is all about data and the numerous formats that defines that data. Sooner or later you will have to be acquainted with these formats. Let's start today.

I understand there is way too many formats but honestly no one remembers all of them. You can always look them up.

We will deal with 4 file formats this week: GenBank, EMBL, FASTQ and MEGA. Here are their description:

EMBL http://www.bioperl.org/wiki/EMBL_sequence_format
FASTQ http://www.bioperl.org/wiki/FASTQ_sequence_format
GenBank http://www.bioperl.org/wiki/GenBank_sequence_format
MEGA http://www.bioperl.org/wiki/MEGA_multiple_alignment_format

We will learn more about this later in the course so don't worry a lot about them. Let's write simple `awk` one liners to check what format our input is in. This `awk` script will be provided only the first line off the file and it will predict what format the file is in.

You can send the first line using head command.

Format recognition can be made based on the first column itself.

The solution can be very clean and simple to something extremely complicated. It's all about how clearly you can think.

Files are provided on T-square for you to check your script.

```
head -n1 <filename> | awk '{if($1 ~ "ID") {print "EMBL file format";} else if($1 ~ "@.*") {print "FASTQ file format";} else if($1 ~ "LOCUS") {print "GenBank file format";} else if($1 ~ "#mega") {print "MEGA file format";} else {print "Did not recognise file format";}}'
```

5.) An *in silico* restriction enzyme digestion

In a parallel universe, restriction enzymes comes by the name of `sed` and breaks microbial genome using substitute patterns. One such enzyme has magically found its way to your computer. Download the M07149.fasta from T-square. We got some cutting to do.

a.) This restriction enzyme works on the pattern `GAATTC` and cuts it right after `G` like this:

Cut the genome into pieces using this restriction enzyme (`sed`)! Store the fragmented genome in a new file. How many pieces did you get? (Don't count by hand. Use something sophisticated like wc)

```
tannishtha@tsom3:Downloads$ sed 's/GAATTC/G AATTC/g' M07149.fasta > M07149restr.fasta
Number of pieces after cutting by restriction enzyme :
tannishtha@tsom3:Downloads$ tail -n +2 M07149restr.fasta | wc -w
303
```

b.) On further investigation, you found that the restriction enzyme is a little flexible. It can actually cut after the first base in the following patterns:

```
GAATTC, GAATTG,
GATTTC, GATTTG,
CAATTC, CAATTG,
CATTTC, CATTTG
```

Update your pattern to cut the genome accordingly. How many pieces did you get this time?

```
tannishtha@tsom3:Downloads$ sed 's/GAATTC/G AATTC/g;s/GAATTg/G AATTG/g;s/GATTTC/G ATTTC/g;s/GAATTG/G AATTG/g;s/CAATTC/C AATTC/g;s/CAATTG/C AATTG/g;s/CATTTC/C ATTTC/g;s/CATTTG/C ATTTG/g;' M07149.fasta > M07149restr1.fasta
tannishtha@tsom3:Downloads$ tail -n +2 M07149rest1.fasta | wc -w
    6801
```

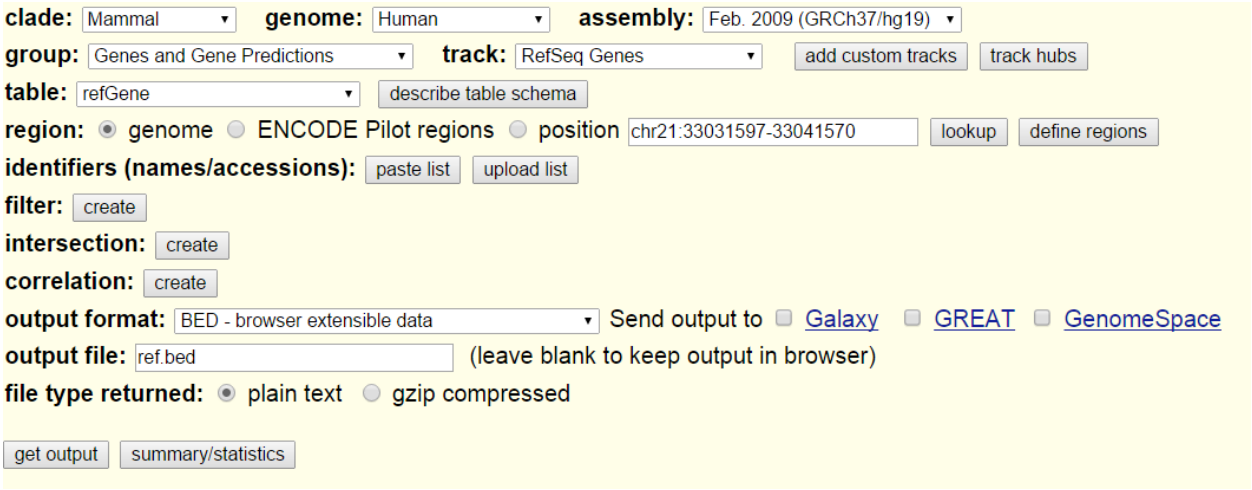
c)You underestimated the strength of this enzyme. It can also vary its length. The updated list of pattern has the following letters optional: third (A or T), fourth (T) and last (C or G). Update the pattern to get the new number of pieces. How many do you get this time?

```
tannishtha@tsom3:Downloads$ sed 's/GAAT/G AAT/g;s/GATT/G ATT/g;s/GATC/G ATC/g;s/CAAT/C AAT/g;s/CATT/C ATT/g;s/CATC/C ATC/g;s/GAATT/G AATT/g;s/CAATT/C AATT/g;s/GAATC/G AATC/g;s/GATTC/G ATTC/g;s/GAATTC/G AATTC/g;s/CAATTC/C AATTC/g;s/GATG/G ATG/g;s/CATG/C ATG/g;s/GAATG/G AATG/g;s/CAATG/C AATG/g;s/GATTG/G ATTG/g;s/CATTG/C ATTG/g;s/GATTT/G ATTT/g;s/CATTT/C ATTT/g;s/GAATTG/G AATTG/g;s/CAATTC/C AATTC/g;s/GATTTC/G ATTTC/g;s/CATTTC/C ATTTC/g;s/GATTTG/G ATTTG/g;s/CATTTG/C ATTTG/g' M07149.fasta > M07149restr2.fasta

tannishtha@tsom3:Downloads$ tail -n +2 M07149restr2.fasta | wc -w
69361
```

6.) Using `awk` to manipulate text on the command line

You should now be somewhat familiar with `awk`. Going over it again, `awk` is a very simple scripting language designed to work with text. It can be used to filter text, format text, gather information about the text, etc. I use it when I want to do something very simple. If I start getting into `awk` scripts that are longer than a few lines, I give up and do it in perl. Use `awk` to do the following with a BED file. Download the **UCSC refGene.txt file** in standard BED format i.e., 12 columns. To do this, go to UCSC genome browser (http://genome.ucsc.edu/). You should see Table Browser in the left panel. Click on it. Select the following options:



Hold on to this original file. I am going to refer to this as **ref.bed**. You can call it whatever you like. I will keep asking you to come back to this. For questions where you have to edit this file, I will always ask you to save the edited version as a new file.

a.) Starting simple, print the unaltered contents of the BED file.

```
tannishtha@tsom3:Downloads$ awk  '$0' ref.bed
$0 means the whole line.
```

b.) Print **every other** line in the BED file.

```
tannishtha@tsom3:Downloads$ awk 'NR%2==1' ref.bed
It prints all the odd numbered lines.
```

c)Print out entries that are on the negative strand only.

```
tannishtha@tsom3:Downloads$ awk '$6 ~/-/' ref.bed
```

d.) For entries on the negative strand, print out the **coordinates** (not the file order) in reverse order – some programs want it like this for reasons I've never really understood. E.g.: if the input looked like this:

```
chr1  25071759   25170815   NM_013943  0     +
chr1  48998526   50489626   NM_032785  0     -
```

I want the output to be like this:

```
chr1  25071759   25170815   NM_013943  0     +
chr1  50489626   48998526   NM_032785  0     -
```

tannishtha@tsom3:Downloads$ awk '{if($6 ~/-/) {chr=$2;$2=$3;$3=chr}}
{OFS="\t";print $0}' ref.bed
Here OFS is the output field separator.

e.) Sum the total length of all of the entries in the BED file, i.e. from start to stop of each entry.

tannishtha@tsom3:Downloads$ awk 'BEGIN{total_length=0}{total_length+=$3-$2+1}END{print total_length}' ref.bed
3006743469

f.) Repeat previous question for those entries on the '+' strand only.

tannishtha@tsom3:Downloads$ awk 'BEGIN{total_length=0}{if($6 ~/+/)
{total_length+=$3-$2+1}}END{print total_length}' ref.bed
1532838735

g.) Change the ID (column 4) of each entry to its order of occurrence in the file, i.e. 0-n.

tannishtha@tsom3:Downloads$ awk '{$4=NR;OFS="\t";print}' ref.bed

h.) Add a new column onto the end of the BED with the value of the new column being equal to the number of columns in the BED. Save this as a new file.

tannishtha@tsom3:Downloads$ awk '{OFS="\t";print $0,NF}' ref.bed >
refextcol.bed
NF=number of fields in an input record.

i.) Change the 'chr' portion of all entries on a single chromosome of your choice to 'chicken', leaving the others untouched.

tannishtha@tsom3:Downloads$ awk '{{if ($1 ~/chr2$/){sub(/chr/,"chicken")}}
{print}}' ref.bed
For substitution, sub function is used.

7.) Let's be awksome!
Download these files from t-square (and course repository
http://jordan.biology.gatech.edu/biol8803b/

if file is not available on t-square)

a.) Count the number of words in file1.txt using awk alone (no wc).

tannishtha@tsom3:Downloads$ awk '{total+=NF}END{print total}' file1.txt
314

b.) Write an awk script to perform FASTQ to FASTA conversion. The FASTQ file is r2.fastq

To perform a FASTQ to FASTA conversion, the beginning @ in the first line of FASTQ file format should be changed to “>”, the second line kept as it is and delete the next two lines in which the third line begins with + and fourth line defines the quality values of the sequence in line2.

```
tannishtha@tsom3:Downloads$ awk '{if(NR%4==1){sub(/@/,">");print}else if(NR%4==2){print} else{next}}' r2.fq > r2chan.fasta
```

c)Write an awk script to perform SAM to FASTQ conversion. The SAM file is map.sam

A SAM file consists of header section which starts with @ and an alignment section. The alignment section has 11 fields of which field 1 is the query sequence name, field 10 consists of the sequence and field 11 has the quality of the associated sequence. Thus these three fields are required for the FASTQ file.

```
tannishtha@tsom3:Downloads$ awk '!/^@/{print "@"$1"\n"$10"\n+\n"$11"\n"}' map.sam > map.fastq
```

d.) Write an awk script to extract reads with flag 12 and output it as a FASTQ file. The SAM file is map.sam

```
tannishtha@tsom3:Downloads$ awk '!/^@/{if($2==141){print "@"$1"\n"$10"\n+\n"$11"\n"}}' map.sam > map1.fastq
```

8.) Merging outputs

`cat` and `paste` aren’t the only commands capable of merging things. There is `&&`

a.) Let’s start simple, merge the outputs of two `echo` commands using `&&`. This can be done as follows: `echo “This is my first line” && echo “This is my second line”`

```
tannishtha@tsom3:Downloads$ echo "This is my first line" && echo "This is my second line"
This is my first line
This is my second line
```

b.) Now let’s get fancy. Print me the first 5 followed by the last 5 lines of ref.bed.

```
tannishtha@tsom3:Downloads$ head -5 ref.bed && tail -5 ref.bed
```

c.) Command inception: Print me lines the 5 lines that comes after the first 5 lines (i.e., only lines 6-10) followed by the 5 lines that comes right before the last 5 lines (i.e., say there are 100 lines, I need lines 90-95). Hint: you will require `( )`.

```
tannishtha@tsom3:Downloads$ (head -10 ref.bed | tail -5) && (tail -10 ref.bed | head -5)
```