Python Workshop 3

Processing Big Text Files

File specifications

The exercises in this workshop are going to focuses on one of three file types described here.

1) Basic pileup file

Each row in the file consists of X columns. The columns in order specify the following information: chromosome, position along the chromosome, reference base, alternate base, quality score, site depth, reads, read qualities.

Here is an example:
```
#CHROM  BASE  REF  ALT  QUAL  DEPTH  READS  READQ
scaffold_1  1867  T  T  30  1  ^=.  E
scaffold_1  1868  C  G  40  4  GGGC E^^^!!A
```

2) Super pileup file

Each row in the file consists of X columns. The columns in order specify the following information: chromosome, position along the chromosome, reference base, alternate base, quality score, SNP quality score, map quality score, site depth, reads, read qualities.

Here is an example:
```
#CHROM  BASE  REF  ALT  QUAL  SNPQ   MAPQ   DEPTH  READS  READQ
scaffold_1  1867  T  T  30  0  28  1  ^=.  E
scaffold_1  1868  C  G  40  20  25  4  GGGC E^^^!!A
```

3) Annotation file

This is the same kind of file you used in homework #1. This is a large list of exons for many genes. The columns are as follows: chromosome, start position, end position, gene name, gene direction (either + or -). Several lines can have the same gene name, that is because the exons specified by those lines are part of the same gene.
Here is an example:
```
scaffold_1      8612153         8612226         PAC:20890324    -
scaffold_1      17270551        17270715        PAC:20891279    -
scaffold_1      17269957        17270295        PAC:20891279    -
```

This lists a single gene with one exon (PAC:20890324) and a second gene with two exons

(PAC:20891279), both of which are on the - strand of the genome.

Basic file processing

In homework 1 you processed a annotation file as described above. Similarly we want to process the pileup file as in filetype #1 above. Say we want to only print out lines from the file that have a quality of at least 30. How would we do this using only a for-loop? Say we need to write another program that prints out only the lines that have a depth greater than 20. Do this.

File processing exercises #1

1. Write a program that goes through a pileup ("pile_old.txt") and prints out only the lines that have a quality of at least 30.

2. Write another program that goes through a pileup and only prints lines that have a depth of at least 20.

3. Write another program that only prints out lines that both have a depth of 20 and a quality of at least 30.

The file format changes...

Well, you wrote all your programs above, and life is good. But, one day you come into work, the program you use to generate your pileup has been updated. Now it prints out some extra columns, and looks like filetype #2 above.

Do your programs still work?

What do you have to do to fix them?

So, when the file format changes, every single program that uses that file is going to break. Wouldn't it be nice if that didn't happen?

File processing exercises #2

1. Fix program #3 from above to now work with the new filetype. ("pile.txt")

Parsing a single line, using an Object

If we *abstracted* the parsing out of our programs, then whenever the file format changes we only need to change one program instead of every single one that uses that file type. Lets build an object, called Record, that hold all of the information from one line. (For now it's just going to use filetype #1 above.) It's behavior is specified below:

```
>>> myFile = open("pile_old.txt", "r")
>>> line = myFile.readline()#first line as above
>>> myRecord = Record(line)
>>> print(myRecord)
Record(chrom = scaffold_1, base = 1867)
>>> print(myRecord.CHROM)
scaffold_1
>>> print(myRecord.BASE)
1867
>>> print(myRecord.REF)
T
>>> print(myRecord.ALT)
T
>>> print(myRecord.QUAL)
30
>>> print(myRecord.DEPTH)
1
>>> print(myRecord.READS)
^=.
>>> print(myRecord.READQ)
E
```

File processing exercises #3

1. Redo the three programs from exercises #1 above using the Record object.

2. Once you've made those programs modify them to work with filetype #2. (Wasn't that easier than changing 3 different files?)

Making iterators

What is an iterator?

What python keyword makes a function into an iterator?

What special python function used in an object will make it into an iterator?

Iterator exercises

1. Write an iterator that steps through and yields each item in a list.
2. Write an iterator that steps through a lit and yields every second item in a list.
3. Write an iterator that steps through a list and only yields the odd numbers from the list.

Making a parser that is an iterator

Lets make a function, that takes a file to parse through, and yields a single Record object for each line:

```
def myMyparser(myFile):
      for line in myFile:
            yield Record(line)
```

Define this function in the same file as your Record class, but not as a function of the class itself.

Explain what each line of code above is doing.

Parser/Iterator exercises and homework

1. You're going to be parsing the annotation file, filetype #3 above. First write an object to hold the information from a single line, then write an iterator that will yield one instance of that object per line.

2. Make a program that goes through all the lines in the annotation file ("annot.txt") and only prints out lines where the exon is at least 500 bases in length.

3. Make a program that prints out the start and end positions of each intron within each gene, rather than the exons.

4. Lets make this a bit more complicated. Usually we care about an entire gene, rather than

just a single exon. Start this by defining a Gene class, that will hold a list of all of the exons, the Gene's chromosome, name, and direction.

5. Once you have a Gene class, make the iterator for the file return an entire Gene at a time, rather than just one exon.

6. Make a program that goes through and prints each gene's name, and start and stop position.

7. Make a program that only prints out the names of genes that span at least 1kb.

8. Define the __iter__ function in the gene class, and make it yield each of the gene's exons one at a time, in order.

9. Make a program that prints out each gene, and tells you the total number of exon that gene has, and the total number of exon that gene has that are more than 500 bases long.