Genomics - Problem Set 1

Due Friday, 1/17/2014 by 9:00am

1. Write a Python script named "array.py" that initializes an array variable with the three amino acids: 'glutamine', 'lysine' & 'proline'. Print out the contents of the array variable. Assign the reversed contents of the array into another array variable and then print that out. Come up with a second way to reverse the contents of the array and print it out yet again. Come up with a third way of solving this problem!

2. Write a Python script that initializes a dictionary with keys that are nucleotide codons and values that are the amino acids encoded by that codon, e.g. 'ATG': 'methionine', 'AAA': 'lysine', etc. (consult http://en.wikipedia.org/wiki/genetic_code or any standard genetics or biochemistry textbook for a complete table). Include all codons for the amino acids from Question 1. Now, use **sys.stdin** to initialize a variable with user input. Have your program print out the codon you entered and its corresponding amino acid by using the variable to index into the dictionary. If you're feeling ambitious, write a program that initializes a dictionary that includes all 64 codons, and save it for possible future use. Save this script as **codon_table.py**.

3. You have carried out a microarray experiment that has identified the location of breakpoints in the yeast genome that are induced under conditions you are interested in. There are quite a few of them, and what you'd like to do is look at genomic features of interest near these breakpoints. As a first attempt, you decide to write a program, **extract_info.py**, that for a **single chromosome** and **coordinate** will print out identifiers of genomic features closest to that breakpoint.

The file that you extract the information about these genes from is:

```
/usr/class/gene211/misc/SGD_features.tab
```

and contains the following tab-delimited columns (* = optional field):

1. Primary SGDID
2. Feature Type
3. Feature Qualifier*
4. Feature Name*
5. Standard Gene Name*
6. Alias* *(multiples delimited by the '|' character)*
7. Parent Feature Name*
8. Secondary SGDID* *(multiples separated by the '|' character)*
9. Chromosome*
10. Start Coordinate*
11. Stop Coordinate*
12. Strand*
13. Genetic Position*
14. Coordinate Version*
15. Sequence Version*
16. Description*

Write a Python script that will examine as a single chromosome and coordinate breakpoint, which will extract information about those genes from **SGD_features.tab**. This breakpoint will be at position **788000** on chromosome **16** - we deal with chromosome 16 below, but you should hard code the coordinate in your script.

Specifically, your script should output the following information for each gene, in **tab-delimited format**, sorted by distance (where distance is calculated from the center of each gene to the breakpoint), with the shortest first:

1. Distance (This should be rounded to the nearest integer as we are working with base pair distances. You can use python's built in `round` function to do this.)
2. Primary SGDID (SGD's database identifier)
3. Feature Name
4. Standard Gene Name

In order to read the **SGD_features.tab** file in Python, we will take advantage of Python's **string.strip()** and **string.split(delimiter)** method while reading the file line-by-line. This input paradigm is a common one while scripting in Python, and your code should look something like this:

```
import sys # necessary for sys.stdin

# other code...

for line in sys.stdin:
        pSGDID, type, qualifier, feature, gene, aliases, parent, sSGDID,
        chromosome, start, stop, strand, pos, cVersion, sVersion, description
        = line.strip().split('\t')

        # code that operates on lines from SGD_features.tab
```

This will split out the tab-delimited (e.g. '\t') fields for you into the named variables above. You can test your script as it develops by only using the first few lines of **SGD_features.tab**

```
head -1 /path/to/SGD_features.tab | python extract_info.py
```

Note that you specify an arbitrary number of lines, e.g. head -2, head -200, etc. By default, **head** pipes the first 10 lines of the input file.

To test your script, you can use the **cat** program as follows (remember, if you don't know what **cat** does, use the **man** command to figure out what it does):

```
cat /path/to/SGD_features.tab | python extract_info.py
```

You can also use redirection to do a similar thing:

```
python extract_info.py < /path/to/SGD_features.tab
```

Note for each open reading frame that there are a minimum of 2 lines, one for the ORF and one for the CDS. In the case that the gene is spliced, there will be more than one CDS, as well as one or more introns. It is most useful to print out **only those top level parts of the feature**. These have a parent feature name that begins with "chromosome", for example "chromosome 16" for our problem. To test where a string begins with "chromosome 16" we will use the **grep** command on the command line to only pipe lines that contain our feature of interest, e.g.:

```
grep "chromosome 16" /path/to/SGD_features.tab | python extract_info.py
```

Next, to re-order the output to list the features closest to our breakpoint first, we'll use the Unix **sort** command (eventually we will learn how to sort directly in Python):

```
grep "chromosome 16" /path/to/SGD_features.tab | python extract_info.py |
sort -n
```

Read the Unix manual page (colloquially known as a "man page" and accessible on the command line via **man sort**) to figure out what the **-n** flag above is responsible for. Next, we only want the nearest 100 features, which we can use our trusty **head** command for once again:

```
grep "chromosome 16" /path/to/SGD_features.tab | python extract_info.py |
sort -n | head -100
```

And finally we are interested in saving our programs output to a file **output.txt**, which can be accomplished using Unix 'redirection':

```
grep "chromosome 16" /path/to/SGD_features.tab | python extract_info.py |
sort -n | head -100 > output.txt
```

This will put all of your results into the file **output.txt**, and this file should look something like:

```
643     S000006329      YPR125W     YLH47
1182    S000006328      YPR124W     CTR1
1231    S000006330      YPR126C
1642    S000006327      YPR123C
...
65623   S000007362      YPR158C-D
67201   S000006298      YPR094W     RDS3
67596   S000007361      YPR158C-C
68009   S000006297      YPR093C     ASR1
```

Optional Work

Since the start and stop coordinates are optional fields, you should ignore those features that don't have those fields since we can't calculate a distance. Can you come up with a way inside your program (or externally via a Unix command) to filter out those features? Hint, many, but not all, of these features have a common 'type' field.

Can you figure you how to limit the results to be just those with centers that fall within 5kbp of our breakpoint?

Submission

You should now have four files for the problem set:

`array.py`
`codon_table.py`
`extract_info.py`
`output.txt` (generated from extract_info.py when run with features from chromosome 16 and breakpoint 788000)

Submit your files just like you did for PS0; put all files in a temporary directory and run the following script:

```
/usr/class/gene211/bin/submit.pl
```