Self-Assessment for Workshop on "Python for Biologists"

A Nanotechnology Platform

Exercise 1: Variables, Operators, Functions, Methods

Mathematical Operations

Molarity Calculator

- Write a script to calculate how much of a compound is needed to solve a given molarity. You will need to create variables to store:
  - The molecular mass of the compound (in g/mol)
  - The volume of solution you want to create (in ml)
  - The desired concentration (molar M)

- The formula will be:
  - Mass (g) = Concentration (mol/L) * Volume (L) * Formula Weight (g/mol)

- Make the script print a summary of the input variables and the calculated value, bypassing these as separate arguments to your print function.

Interactive Molarity Calculator

- Make a modified version of your initial script in which the different values you need for the calculation are requested interactively using input statements.

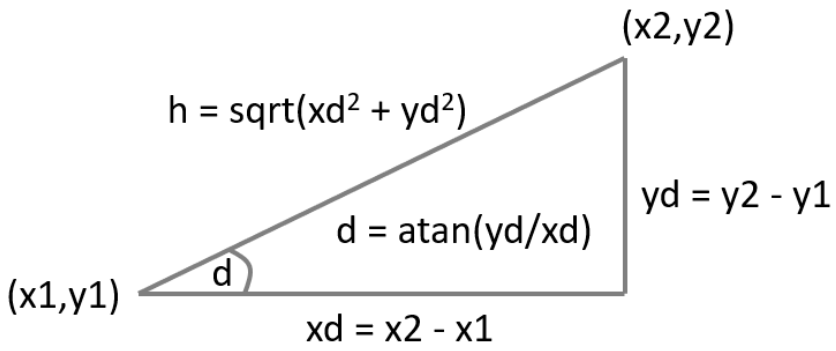
- Remember that the return value from the input will be a string (str), which you can't use in mathematical calculations. It would be best if you used float() to convert it to a numerical value.

- So that the calculated value doesn't get too specific, use the round function to limit the precision to 2 decimal places when printing the result.

2D Geometry Calculator

- Make a script which defines two 2D positions (x1, y1, x2, y2) and then calculates the distance between them (Pythagorean distance). Just make up some coordinates and put them in your script.

- Also calculate the angle (in degrees) from the first point to the second



- You will need to use the sqrt and atan functions from the math package; the angle will be in radians, so you can use the degrees function from math to convert this. https://docs.python.org/3/library/math.html

String Manipulation

Organism names

- Write a script to correctly format organism scientific names (e.g. Homo sapiens).
    - It should interactively ask for the Genus (e.g. homo) and Species (e.g. sapiens) using input() statements
    - It should then print out the name with the Genus with an uppercase initial letter and the species all in lowercase.

- Look at the capitalise and lower methods in the str class to do these transformations. https://docs.python.org/3/library/stdtypes.html#textseq

Text alignment

- Write a script which uses five input statements to ask the user for five names and stores them in 5 separate variables
- After collecting the names, print them out, one per line, using the centre method to centre them within a 40-character block
  - Make sure that the names are all in uppercase (upper method), and make sure that any spaces have been removed from the ends of the names (strip method)

- At the bottom of the list, add the total length of all the entered names.
  - You will need to use the len function to get the lengths of the names, which you can then add together.

Tricky challenge (if you have time/motivation!)

- Make a modified version of your 2D distance calculation script, except that this time, the user supplies the following information
  - First x position
  - First y position
  - Distance to the second point

- Your script should then calculate an x/y position which would fall that distance from the first point. Since many issues would fit these criteria, you must use a method from the random package to select a suitable arbitrary point.

Your script should print out the final result and explain its working.

Exercise 2: Data Structures

Lists and Dictionaries

Simple Data Collection

Write a script that prompts the user to enter five numeric values using five input statements. Convert these to numbers using float() and put them into a list. Use the sort method to order the list, then print it.

Use a list selection using [ ] to pull out and print just the list's first (lowest) value.

Next, reverse the sorted list and print it again to see the numbers from highest to lowest. Look at the documentation for the sort method to know how you could have done the sorting this way initially.

Finally, print out a statement which uses the count method to say how many times the number 2 was present in the data.

Animal Sets

Write a script which asks the user to input the names of 5 animals. Put each of these into a set. Then, use the in-test to ask them for final input and print out a message saying whether the animal they entered last has been seen before. Make the query case insensitive by converting everything to lowercase.

For the output, you can print True or False – we'll be able to do more apparent things with the data after next week's session.

Random Data

Make an empty list and then populate it with ten numbers drawn from a normal distribution with a mean (mu) of 10 and a standard deviation (sigma) of 3

Calculate the mean, standard deviation and standard error of the mean of the simulated data.

You can import random and then use the unexpected—normalvariate function to get the random values.

You can import statistics and then use statistics. Mean and statistics. stdev to calculate those values. For the SEM, you must divide the SD by the square root (math. sqrt) of the number of data points (len of the list) minus one.

Random Genomic Position

Write a script to generate a random genomic position of chr3:+:10200353, i.e. chr:strand: position.

Create a list of chromosome names and use the random. Choice function from the unexpected package to select a random item from the list.
Create a dictionary with chromosome names as keys and their lengths as the values. Use this to get the size of the randomly selected chromosome, then use the random—randint function from the unexpected package to choose a random integer smaller than that value.
+
Finally, create a tuple of + and - and use random—Choice to select one of those values.

Print all of the randomly selected values you created.

Multi Level Data

Amino Acid Properties

Make up a data structure of the properties of amino acids. The top level should be a dictionary where the keys are the one-letter amino acid codes, and the values are another dictionary with keys of "long name" and "molecular weight" containing the appropriate discounts. Details of the values can be found here.

Allow the user to enter a one-letter amino acid code and have the script print the long name and weight for that amino acid.

Gene Model

Put the information below into a suitable multi-level data structure. It makes sense to build the top-level gene information in one go and then add the individual transcripts afterwards.

Gene			
Name	Nanog		
	Nanog		
Description	homeobox		
Location	**Chromosome**		12
	Start		7,787,794
	End		7,799,146
	Strand	Forward	

Transcripts	**Name**	**ID**	**Length**	**Amino Acids**
	NANOG-201	ENST00000229307.9	5182	305
	NANOG-202	ENST00000526286.1	870	289
	NANOG-204	ENST00000541267.5	836	186
	NANOG-203	ENST00000526434.2	558	0

The top-level structure will be a dictionary. The Location value will be a second dictionary containing the different pieces of location information.

The Transcripts value will be a list; each will be a dictionary containing the different transcript details.

Use this data structure to pull out the length of the last transcript.

Reconstruct a location string by querying the structure.

Tricky challenge (if you have time/motivation!)

Create a program to simulate a random sequence based on the composition of a reference sequence.
The inputs to the program should be

- A reference DNA sequence
- The length of a random sequence to simulate

You can treat a string such as "GATATCG" as a list of many operations for reasons we'll come on to.

Make your script count and report the occurrences of each of the four letters. Print out a count for the number of letters not accounted for by the four expected bases.

Use the random. Choices function to select the requested size from a population, which reflects the composition of the original sequence. Since your original series may have non-GATC letters, construct a list of choice weights.

Your script should cope with the input sequence being uppercase, lowercase or a mix of the two. It should force the length to be an integer.

Exercise 3: Iteration and Conditions

Iteration and Looping

Gene Set Intersection

Make two lists (or tuples, if you prefer) of gene names containing:

List 1: "Npepl1", "Rab13", "Reg4", "Asb17", "Clcnka", "Nup62", "Upf3a", "Kcnn1", "Ccdc151", "Arg1", "Tmem98", "Mtx3", "Isl1", "Fam53c"

List 2: "Kcnj2", "Rab13", "Reg4","Nol6", "Masp2", "Clcnka", "Upf3a", "Kcnn1", "Arg1", "Krt75", "Smpd3", "Mtx3", "Trim8", "Fam53c"

Write a script to build a new list containing the genes in List 1, also present in List 2.
Create an empty list which will store the genes you select. Write a for loop to go through the genes in the list. For each one, use an in-test to see if it's in list2 and append it to the new list if it is.

When writing out the results, sort the intersection genes alphabetically and number them (using enumerate) as you print them out, i.e.:

1. AAC
2. BCL

Etc.

Angle Ranges

Find which ones have math for the set of integer angles from 0-1800 degrees— sin value of zero. Use a range statement to create a for loop through the grades.

Because you will be calculating small fractional values, an == equality test is unreliable, so test for the absolute (abs function) sin value being less than 0.01.

Variant Counting

You are running an experiment looking for a series of protein variants, specifically.

E23A, P12S, W88Y and R32N

Write a script which repeatedly prompts you to enter a variant name and keep count of the number of times the ones in the list come up. If you get a variant that isn't on the list, print out a warning and move on.

If they enter a blank variant, stop asking for more input and print out the counts of the variants you saw.

To do this:

- Create a dictionary with the variant names as keys and an integer (initially 0) as values
- Start an infinite loop with while True:
- Ask for the name of the variant in an input statement.
- If nothing was entered, then break to exit the loop
- Look up the variant in the dictionary and increase the value by using += 1
- Print out the dictionary after you exit the loop.

Molecular Weights

Use the amino acid properties data structure you made in Exercise 2 and calculate the total mass of the antigenic peptide:

'T', 'E',' N', 'K',' Y',' S', 'Q', 'L', 'D', 'E', 'E', 'Q', 'P'

Log transformation

Using nested lists, create a random 2D dataset of 10 rows and 10 columns. Populate it with random values between 1 and 1,000,000.

Start with an empty list, then use two nested for loops using range(10) to append a new list and append math—randint () values into it.

Log transforms the data by iterating the original data structure and building a new one. Again, this will be two loops, one to go through the ten lists and the second to go through the ten values in each index. Reduce the precision of the log-transformed values to 1 decimal place using round.

Print out the transformed data.

All hexamers

Write a program to collect and print out all 6-mer combinations of the DNA bases GATC.

There are multiple ways to do this. One would be to make a list of the bases, add each base to each position over six rounds and replace the original list. To append a letter to a string you use the + operator, so "GA" + "TC" = "GATC"

Tricky challenge (if you have time/motivation!)

Create a data structure to hold the following genomic positions from the human GRCh38 genome.

Chr1:90435481-90535480
Chr4:121080701-121180700
Chr5:58203396-58303395
Chr6:24011285-24111284
Chr7:27397324-27497323
Chr9:63677076-63777075
Chr12:57831538-57931537
Chr13:80438618-80538617
Chr16:86177236-86277235
Chr18:39459388-39559387

Use the code you wrote in last week's exercise to generate random genomic positions; only this time, use a loop to create jobs and check them to see if they fall into any of the list of regions above. Count how many times you hit each area. Stop when any of the parts has ten hits in it.

Finish by printing the number of hits you got to each region and the number of random positions you had to generate to achieve this.

Exercise 4: String Manipulation

String manipulation

Mappability Presentation

Below is some data from calculating the mappability of some genome regions for sequencing reads of different sizes.

From the data, calculate the percentage mappability (mapped fragments as a percentage of total pieces), and then write out the results as nicely formatted text using f-strings.

- The percentage values should be shown to one decimal place
- The different columns should line up (be the same width in all lines)

Read Length	Fragments	Mappable
30	1725	697
40	1713	794
50	1689	912
70	1677	1103
85	1660	1187

To load this data, initially copy it into a triple-quoted string in your script so you can have multiple lines. Use the split method to split on newlines (\n) to get a list of sequences. Use a for loop to iterate through the ropes to split out the different values and put them into a suitable data structure.

You can use f-strings to format the lines as you print them.

Data Interpretation

You have been given a set of dates which are in inconsistent formats.

2018-04-16 10:25:29 AM MDT
2018-01-05 12:05:00 PM CST
Tue Jan 02 2018 08:12:10 GMT-0600 (CST)
2018/01/01 4:50 PM HST
01/01/2018
12/02/2018

Write a script to parse these and produce a tuple of (year, month, day) as integer values. There are ways to achieve this with standard string parsing and regular expressions. One process would be:

- Check the first letter of the date. If it's numeric, parse it as a delimited number, otherwise parse it as text

- For the delimited numbers, extract everything up to the first space. Check for a minus or slash to confirm the delimiter and split the data into sections. Check the length of the first section to see if you've got day-month-year or year-month-day

- For the text, dates are split into sections based on spaces and then extract the day, month and year from those. Use a dictionary to map text months to numbers

When printing out results, use a format string. You can use the str. fill method to add leading zeroes to a value to get consistent formatting of days and months.

Regular Expressions

Cleaning Sample Names

You have been given some files with embedded sample names

lane1_NoCode_L001_R1.fastq.gz
lane1_NoIndex_L001_R1.fastq.gz
lane1_NoIndex_L001_R2.fastq.gz
pipeline_processing_output.log
lane7127_ACTGAT_JH25_L001_R1.fastq.gz
lane7127_ACTTGA_E30_1_2_Hap4_24h_L001_R1.fastq.gz
lane7127_AGTTCC_JH14_L001_R1.fastq.gz
lane7127_CGGAAT_JH37_L001_R1.fastq.gz
lane7127_GCCAAT_E30_1_2l_Hap4_log_L001_R1.fastq.gz
lane7127_GGCTAC_E30_1_4_Hap4_48h_L001_R1.fastq.gz

We want to extract the sample name from these files. The structure is:

1. Written lane number
2. Barcode
3. Sample name
4. Numeric lane number (starting with L)
5. Read number (R1/2/3/4)
6. File suffix (always .fastq.oz)

So for lane7127_GCCAAT_E30_1_2l_Hap4_log_L001_R1.fastq.gz, the sample name would be
E30_1_2l_Hap4_log

Extract the sample names from the file names, ignoring any files which do not match the expected pattern.

You will need to use a combination of basic string processing to do some simple checks on the files, with more complex regular expression substitutions to remove the unwanted parts of the filename from the start and end of the string to leave just the sample name in the middle.

Tricky challenge (if you have time/motivation!)

Transcription Factor Binding Sites

Write a program to search a DNA Sequence for the presence of one of the following transcription factor binding sites, which utilise ambiguity codes. In each case, write a regular expression to represent the binding site and then search for all of the positions of that site in the sequence being searched.

Code	Represents
A	Adenine
G	Guanine
C	Cytosine
T	Thymine
Y	Pyrimidine (C or T)
R	Purine (A or G)
W	weak (A or T)
S	strong (G or C)
K	keto (T or G)
M	amino (C or A)
D	A, G, T (not C)
V	A, C, G (not T)
H	A, C, T (not G)
B	C, G, T (not A)

Transcription Factor	**Consensus Sequence**
RUNX1	BHTGTGGTYW
TGIF1	WGACAGB
IKZF1	BOGARD

The sequence to search against is shown below. Enter this directly as shown, and extract a single string of DNA bases from the structure you were initially given (i.e. remove the header and the line breaks).

>search_seq
GACACCTCAGTACTAGGATGNNNNNNTATCAGCCTGAACTAGCAGGCCTGGTTCC
AAATT
TTTTTATCAACACTCGTAGGGGGATTATCCTAGAGGGGGTCTGGGATTTCTTTGA
CATCA
GAGTATTTTTGCCTTGCTCCTTCACAATTTGGGAACAAATAATTTAGTGGTTATT
AACCC
TGGCTACGCACTGGAAACTTTAAAAATAATGCTGGTATGAAATTTACACAGAGTA
TCGTG

AAAATTTTCACTGAGTACCATGTGGTTATACATTGGATAAGGCTCCAGGAAGCAG
CTACT
GGAAGACAGCCATGCCAAGAGTGGTTAGTGGTTGGAATTTTGGCAAGTCAGTTTT
AGTCT
GCCTTATCAAATACATGGGCATACAGATAAATCCTTAGATGGCTCTCCTACTTAC
TGAAA
CATTTTCTATCTATCTATCTATCTATCTATCTATTTGGGAAGCTATCTATCTATCT
ATCA
TTTATTTAAGGTAGTCTCTATCTGCCTCTGTCTCTGTCTGTCTCTGTGTCTCTGTG
TCTG
TCTGCTCTCTCTCTCTCTCTGTGGGAATCTCTCTCTGTGTGTGTGTGTGTATGTGT
GTGT
GTGTGTGTGTGGTGTGCATGAACATGAGTAAAATCCATAAGGAAACTTTCAGAGT
TGGTC
CTCTCCTTATATCAAATGGATCCAGGAATTAAACTCAGGTTCAATTCTTGGTGCCT
TTAC
TAGTTGAGCCATCTCACTGGCTCTTCATCATCTTTAGAATAAACTCACTTTATTAC
ACAC
ACACACACACACACAACCTGGGAGTACACACACACACACAACCAAAGCCCCAACGG
AAAA
CTACAATATTATAATGAATACACAGGTTCTCAACATAGTCTCTGCCACGCTTGCA
GACAA
AGATGAGTAGAAGTAGAAAGAACCAGGGAAACGTGGAGCAAGTCAGAAGGAATA
ACAGTC
AGAAGGAATAACAGTCAGAAGGAATAACAGTCAGAAGGAGTAACAGTCAGAAGG
AATAGC
AGTCAGAAGGAATAACAGTCAGAAGACAGCACAGTCAGAAGGAATAACAGTCAGA
AGGAA
TAACAGTCAGAAGGAATAACAGTCAGAAGGAATAACAGTCAGAAGGAATAGCAG
TCAGAA
GGAATAACAGTCAGAAGGAATAACAGTCAGAAGGAATAACAGTCAAAGAAATAG
CAGTCA
GAAGGAATAGCAGTCAGAAGGAATAACAGTCAAAGGAGCAGTCAGAAGGAGTAA
CAGTCA
GAAGGAATAACAGTCAGAAGGAATAACAGTCAAAGGAATAGCAGTCAGAAGGAG
TAACAG
TCAGAGCAAACACAGAGATGACAAAGGCAATGGGGTCAGAGACTTCACCACTCTC
CAAGA

You will need to use the re. Finder function to search for multiple hits and return an iterator of re. Match objects, on which you can use the span() function to get a tuple of the start and end points for each match.

Exercise 5: Reading and Writing Files

You will need to download and uncompress the zip file of the data files we have provided for these exercises.

Filtering Data Files

Simple Filtering

Read the data in the cancer_stats.csv file. This is a dataset of case and fatality numbers for different types of cancer, split between males and females.

We want to extract from this file the subset where the survival rates for females ((cases-deaths)/cases) were higher than for males. We will exclude any cancer types only one sex can get (values are NA for the other sex).

This is a CSV file, so you can read it and split it into commas. You could use the CSV package to read for a more complete solution.

Initially, you can print the results to the screen, but you could also try saving the results to an output file, preserving the headers from the original data.

Matched Filtering

You've been given two files:

1. statistical_hits.txt is a set of p-values and FDRs (corrected p-values) for a set of genes
2. genome_annotation.txt is a list of genome annotations

Write a script to add the genome annotation to the statistical hits and write the results to a new file.

Iterating over files

Mapping data extraction

In the Mapping Stats folder, you've been given a set of 126 mapping results files from a large sequencing dataset where each file has a name like:

ERR2588244_Mbd3FLAG_Chd4_R1_GRCm38_bowtie2_stats.txt

..and has contents which look like:

```
211539334 reads; of these:
  211539334 (100.00%) were unpaired; of these:
    5252917 (2.48%) aligned 0 times
    151453128 (71.60%) aligned precisely one time
    54833289 (25.92%) aligned >1 times
97.52% overall alignment rate
```

Write a program to iterate through these files and collate the percentage of reads aligned exactly once (71.60% in the above example). Print out the file name, removing everything from _GRCm38 onwards and the extracted mapping percentage.

Finding Files

Write a script that finds all Excel files (.xlsx) under a specified starting directory. Have it search through the files in your home directory (or wherever you might have some Excel files). For each file, print its location and its size.

Tricky challenge (if you have time/motivation!)

In the Bacteria fold of the course data are a series of gzipped GTF files. These are the genome annotations for a series of different bacterial species. The GTF

format is described here: https://www.ensembl.org/info/website/upload/gff.html.

Iterate through these files and read them using the gzip package; for each species, count the number of genes present in the organism, the length of the most significant gene, and its name.

Exercise 6: Writing a complete application

Only one exercise this week, but it's a more significant task than the previous exercises you've been given.

We want you to write a complete end-user application called tf_search.py

This program will take in the location of a DNA sequence in fasta format and the name of an output file and will search for transcription factor binding sites in the input sequence file and report them into the output file. By default, the program should search with all of the transcription factors it knows about, but the user should also be able to specify a specific sub-list of transcription factors to use.

We have provided you with a list of human transcription factors containing their (ambiguous) consensus sequence in the file Transcription Factors/human_tf_consensus. Tv. You can either read this from wherever you have your data files unzipped, or you can put it alongside the script file and use the __file__ particular variable to find the location of the current script so you can find the associated data file.

You must split your program into functions and call them sequentially from an initial primary part. The script you write should also be able to be called as a library. One suggested order of operations would be something like:

```
def main():
    options = read_options()
    sequence = read_sequence(options.sequence)
    tf_list = read_tf_list(options.tfs)
    hits = search_for_tfs(sequence,tf_list)
    print_hits(hits,options.outfile)
```

You may also want to write other functions to handle tasks within one of the top-level functions (for example, generating a regex from a consensus sequence). Use whatever structure you think is best.

Each function should have some essential documentation allowing the help function to work with them.

- You should use the argparse package to manage the command line options. It's up to you how you choose to implement the required options.

- The sequence will be in standard Fasta format, and you should record both the sequence name and the contents of the sequence

- The TF list is a TSV file, so you can split each line based on tabs to get the individual pieces of data. You only need the name of the factor and the consensus sequence from the file.

- Your output file should contain the following information for each hit.

    1. The name of the sequence
    2. The name of the transcription factor
    3. The start and end position for the match
    4. The consensus sequence for the transcription factor
    5. The actual sequence matched during the search

- Your output file can be either a CSV file or a TSV (tab-separated value) file

We have given you a sequence for the scyl3 promoter, which you can use to test the program. If you want to try specific factors, you should see hits in this sequence, including impacts to FOXK1 and ISL1.

Tricky challenge if you have time

Some extensions to the essential exercise if you want to try more advanced options. We have provided a gzipped version of all of the promoters on human chromosome 1 to give you a larger dataset to work with.:

- Write a test suite to go with your program and validate the basic functionality of the program
- Make your script also support reading from gzip compressed multi-sequence fasta files
- Make your script support searching on the reverse complement strand
- Add an option to ignore TFs with more than *n* matches, where *n* is an option
- Add progress messages so you can see what the program is doing, but add a --quiet option to suppress them.

Exercise 7: Writing an application using external resources

You will write another user-facing application using external API resources for your final exercise.

The application will take in one or more human gene names and write out a file containing the details of the existing transcripts for those genes. The information for this will come from the Ensembl REST APIs (https://rest.ensembl.org)

To connect to Ensembl, you will use the requests package, which is not part of the standard library, so you must install it using pip.

Later, you will use the Python package to parse a fast format sequence file. You will need to install Python, which is also available from PyPi.

Inputs

Your program should be able to run in either an interactive or non-interactive manner. In the non-interactive mode, you should be able to specify the list of genes on the command line, as well as the name of an output file to write to, e.g.:

python3 gene_query.py --genes Nanog Sox2 Brca2 --out file gene_results.txt

The program should run without further user interaction and write the results to the output file. You can emit progress messages, but these should be sent to sys. Stderr, you should be able to suppress these by adding a --quiet option to the command line. For the genes option, you must add nargs="+" to the argument parser add_argument call to specify multiple values.

In the interactive version of the program, you would be able to omit the gene names:

python3 gene_query.py --out file gene_results.txt

You would be prompted to enter as many gene names as you like in the terminal (one gene at a time).

E.g.:

python3 gene_query.py --out file gene_results.txt
Which gene? Nanog
Which gene? Sox2
Which gene? Brca2
Which gene?

..after entering a blank gene name, the program should continue.

API queries

You must use two different API queries to get the desired results. The first will take in a gene name and return the Ensembl ID of that gene. If multiple genes match the exact phrase, take the first one. The API you need is this one: https://rest.ensembl.org/documentation/info/xref_external.

You can query against the HGNC database, which defines the gene names.

Once you have the gene ID, you can get the transcripts' details for that gene. You are going to need to get the transcript ID and the sequence. To do this, you can use this API: https://rest.ensembl.org/documentation/info/sequence_id. The type of query will be cDNA, and you will also need to set multiple_sequences=1 to allow multiple transcript sequences to be returned for a single gene ID.

Sequence Parsing

Once you retrieve the sequences, you can get them as a string from the request object using r.text. The series will contain multiple lines in a fast format. Rather than parse this yourself, you can use the SeqIO package from biopython: https://biopython.org/wiki/SeqIO. Since the input to SeqIO needs to be a stream (i.e. filehandle) rather than a string, you need to use the io.StringIO package to create a stream from the line you get back: https://docs.python.org/3/library/io.html.

Calculated Values

Calculate its length and GC content once you have extracted the sequence from the parsed fasta file.

In the output file, the fields you need to record are:

1. Gene name (whatever the user provided initially)
2. Gene ID (the Ensembl ID you retrieved from the API)
3. Transcript ID (from the second API query)
4. Transcript length
5. Transcript GC content

A query for Nanog, Sox2 and Brac2 should look like:

gene_name	gene_id	transcript_name	length	gc
Nanog	ENSG00000111704	ENST00000541267.5	836	49.8
Nanog	ENSG00000111704	ENST00000229307.9	5182	44.7
Nanog	ENSG00000111704	ENST00000526434.2	558	45.3
Nanog	ENSG00000111704	ENST00000526286.1	870	49.0
Sox2	ENSG00000181449	ENST00000325404.3	2512	50.7
Brca2	ENSG00000139618	ENST00000544455.6	11854	35.8
Brca2	ENSG00000139618	ENST00000530893.6	2011	38.2
Brca2	ENSG00000139618	ENST00000380152.8	11954	36.2

Brca2	ENSG00000139618	ENST00000680887.1	11880	36.0
Brca2	ENSG00000139618	ENST00000614259.2	11763	35.8
Brca2	ENSG00000139618	ENST00000665585.1	2598	40.7
Brca2	ENSG00000139618	ENST00000528762.1	495	41.0
Brca2	ENSG00000139618	ENST00000470094.1	842	41.4
Brca2	ENSG00000139618	ENST00000666593.1	523	39.6
Brca2	ENSG00000139618	ENST00000533776.1	523	40.0