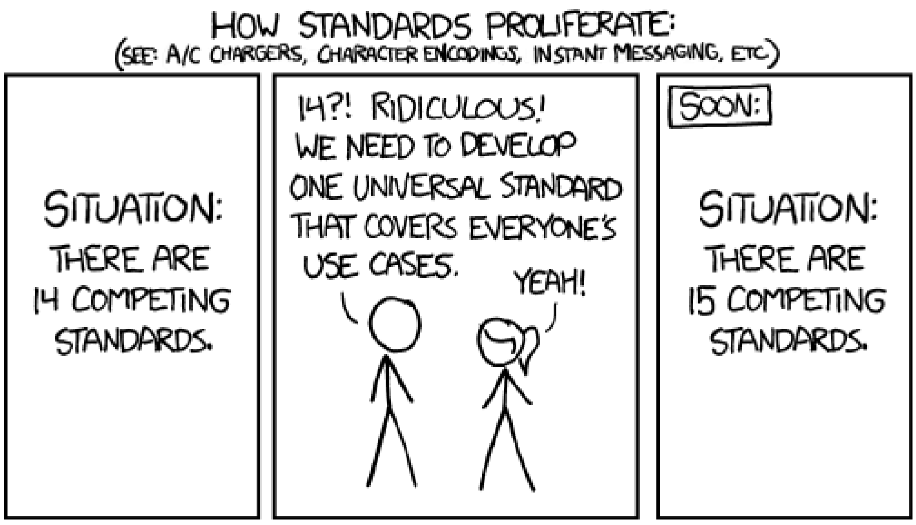
HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Link: https://xkcd.com/927/ (Creative Commons Attribution-NonCommercial 2.5 License)

Home assignments

September 24, 2019

Contents

Guidelines

Instructions

- All the hand-in exercises in this course will require you to write Python programs.
- We do not require that you write any reports describing the programs, but we do require you to comment your code in a special way so that we can follow how you have been thinking.
- The program must start with a comment section which describes the general outline of the code, followed by a description of the procedure used.
- We strongly recommend that you start by writing this comment section **before** you begin to write the code.
- The comment block should contain the following headings:

  Title

  Date

  Author

  Description

  List of user defined functions (if any)

  List of modules used that are not explained in the course material

  Procedure

  Usage

Here is an example:

```
#!/usr/bin/python3

'''
 Title: prime.pl
 Date: 2017-10-07
 Author(s): Björn Canbäck

 Description:
    This program will output the first n numbers of prime
       numbers.
    What n to use is decided by the user.
    The script uses the input method so the user will
       interactively be asked what number to use.

 List of functions:
    No user defined functions are used in the program.

 List of "non standard" modules:
    sh: Can call any external program as if it was a
       function.

 Procedure:
    1. Get the number of prime numbers to use from STDIN.
    2. Iterate over all integers from 2 to an unknown number.
    3. Test if the integer is a prime number. If so, add it
       to the list of prime numbers.
    4. When the list contains the numbers of prime numbers
       to produce, end the iterations.
    5. Print all elements in the list, each item on one line.

 Usage:
       ./prime.pl number_of_prime_numbers_to_report
'''

#################################
```

Rules for the assignments

There are a few rules for the assignments:

- The three assignments are mandatory, you need to get a "pass" on each of them in order to pass the entire course.
- For some of the hand-in exercises there are sample input and output (result) files. When running your Python program you should get "similar" and sometimes exactly the same results.
- Before you hand in the exercises make sure that:
  1. the program contains a documented program header.
  2. the program runs without errors and warnings.
- You should all hand in (by mail) individual solutions, but it is of course allowed (and even recommended) to ask questions, to discuss etc. with other students. Again, it is not allowed to copy code from another student. When we mark the solutions we will also compare these between students. Of course parts of the code will be identical between students, but we take a look at the overall impression.
- Have fun!

Mailing list

The assignments should be mailed according to the following list:

- Assignment 1: Jakob Willforss and Björn Canbäck
- Assignment 2: Aaron Scott, Björn Canbäck
- Assignment 3: Björn Canbäck

Latest submission dates:

- Assignment 1: Oct. 16
- Assignment 2: Oct. 21
- Assignment 3: Oct. 25

Home assignment 1

Data for the assignment can be downloaded from *live@lund*. The name of the archive file is *homeAssignment1.tgz*. A common task for bioinformaticians is to merge data from several files into one. Your task is to insert information about the protein description (called *hitDescription*) acquired from a `blastx`-run using the gene sequences as queries and the `UniProt` database as target into a fasta file.

The fasta file named `malaria.fna` looks like:

```
>1_g    length=1143     scaffold=scaffold00001   strand=-
ATGGATATAAATAAGAAATATTTTGCTCAAGATAAATTGGAACATCACTATCACCA...
>2_g    length=531      scaffold=scaffold00001   strand=+
ATGTTAAAATTTGTAAATATGATTCAGTTATTATCAAGAGGTCACAAACTTGTGGAA...
...
```

Note that the fields in the id lines are tab-delimited.

The blastx output file named `malaria.blastx.tab` looks like:

```
1_g 1143 70 1140 P14223 369 13 369 +1 Fructose-bisphospha...
2_g 531 1 489 Q8IL24 172 1 159 +1 Putative uncharacterize...
...
```

Note that the fields in the lines are tab-delimited.

The output file should look like:

```
>1_g length=1143 scaffold=scaffold00001 strand=- protein=Fructose...
ATGGATATAAATAAGAAATATTTTGCTCAAGATAAATTGGAACATCACTATCACCA...
>2_g length=531 scaffold=scaffold00001 strand=+ protein=Putative...
ATGTTAAAATTTGTAAATATGATTCAGTTATTATCAAGAGGTCACAAACTTGTGGAA...
...
```

Note that the fields in the id lines should be tab-delimited. Use the entire protein description (not included here due to lack of space). If the gene doesn't have a blastx hit, indicated by `null` in the blast-file, it should *not be* *included* in the output file.

Name the program `malaria.py`. Mail it together with a compressed output file called `malariaDescription.fna.gz`.

Home assignment 2

Data for the assignment can be downloaded from *live@lund*. The name of the archive file is `homeAssignment2.tgz`.

We want to calculate how well two DNA sequences align to each other:

```
>id1
a  c  g  t  a  g  g  t  t  t  t  a
>id2
a  t  g  t  -  -  g  t  t  t  g  a
```

If we have a match between two nucleotides in a given position it will be scored +1. If we have a transition, `a<->g` or `c<->t`, the penalty will be -1 and if it is a transversion (any other substitution) the penalty will be -2. A gap gives the score -1. A figure showing transitions/transversions is found here:

https://en.wikipedia.org/wiki/Transversion#/media/File:Transitions-transversions-v4.svg

The score for this alignment is 3:

```
a   c   g   t   a   g   g   t   t   t   t   a
a   t   g   t   -   -   g   t   t   t   g   a
1  -1  +1  +1  -1  -1  +1  +1  +1  +1  -2  +1  =  3
```

Write a program that outputs just one value, the alignment score. The program should be run as:

```
score.py alignment_fasta_file
```

The input fasta file used for testing and in this example, `score.example.fna`, is found in the archive together with the file `score.fna`. Using your program, what alignment score do you get for the alignment in `score.fna`? Mail the answer together with your program that should be named `score.py`.

Optional complication, do it only if you are interested

First two definitions:

gap opening penalty The cost of opening a gap.

gap extension penalty The cost of extending the gap. The penalty is for each extended gap position.

We calculate the score for two aligned sequences:

```
a  c  g  t  a  a  g  g  t  t  t  t  a
a  t  g  t  -  -  -  g  t  t  t  g  a
```

If we assume a match is rewarded 1, the transition cost to be -1, the transversion cost to be -2, the gap opening penalty to be -2 and the gap extension penalty to be -1, the score will be 1:

```
a   c   g   t   a   a   g   g   t   t   t   t   a
a   t   g   t   -   -   -   g   t   t   t   g   a
1  -1  +1  +1  -2  -1  -1  +1  +1  +1  +1  -2  +1  =  1
```

The c-t transition scores -1, the gap is first penalized with -2 when opening and then with -1 twice since it is extended. Finally the t-g transversion scores -2.

Note that since the program takes a multiple alignment as input, that there may be positions in the pairwise alignment where you can have two gaps at the same site. Score this as 0.

Write the program so it can take any number of sequences. Let the user decide (see example below) the transition cost (-1), the transversion cost (-2), the gap opening penalty (-2), the gap extension penalty (-1) and the match score (1). The arguments should be ordered in this way. The program should output a tab-delimited upper triangular matrix:

```
     id2   id3   id4
id1  44    20    35
id2        47    20
id3              25
```

Note that the matrix values are hypothetical and that you in fact can get negative scores. Send your program `score.extra.py` (you can always submit `score.py` as well) together with the output matrix that is output when running the program on the file `score.extra.fna`. Example run:

```
score.extra.py -1 -2 -2 -1 1 input_fasta
```

Home assignment 3

Data for the assignment can be downloaded from *live@lund*. The name of the archive file is `homeAssignment3.tgz`.

1. Write a program that prints (absolute) abundances for each amino acid to a file. Input is a multi fasta file with possibly interleaved sequences. Only the 20 standard amino acids should be printed, otherwise denote the letter as X. Example input:

```
>id1
ACQWRDB
FHIGLSB
>id2
ACDWRDJ
WHIGLSJ
```

   The program should take one argument, the fasta input file. Output should be ordered and would in this case look like:

```
A 2
C 2
D 3
E 0
F 1
...
W 1
Y 0
X 4
```

   The two fields should be separated with a space, not a tab. The script should handle both lower and upper case amino acids. The 20 amino acids have the abbreviations:

```
A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V,
    W, Y
```

The script should run like:

```
amino.py amino.faa amino.list
```

Name the program `amino.py` and mail it together with the output file `amino.list` after running the file `amino.faa` as input. Remember that you can often double check at least parts of the results in `bash`.

2. A common procedure before sequencing is to put a barcode (a defined oligo nucleotide) in the 5'-end of the sequences (it can also be used in the 3'-end). The barcodes often represent different samples. You are provided with the fastq file `barcode.fastq`. Filter the file on barcodes (exact matches only) and print each sample represented by the barcode to it's own fastq file after removing the barcode. The barcodes are `TATCCTCT`, `GTAAGGAG` and `TCTCTCCG` and will be found in the 5'-end only. The corresponding sample names are `sample1`, `sample2` and `sample3`. The output files should be named `sample1.fastq`, `sample2.fastq` and `sample3.fastq`. Not all sequences will contain these three barcodes. Save these to the file `undetermined.fastq`.

   Tip: More than one file can be opened with the same `with` statement:

```
with open('1.fna', 'r') as in, open('1.faa', 'w')
   as out:
```

   For clarity this may be written as:

```
with open('1.fna', 'r') as in, \
     open('1.faa', 'w') as out:
```

   No white space is allowed after the backslash.

   Double check at least parts of the results in `bash`. Name the program `barcode.py`. Mail it together with a compressed `sample3.fastq` file.