BPSM ICA2: A Python3 workflow

Due date: 1400 GMT, Mon 20 November 2023

Image source

Tasks overview
Write a generic Python3 programme/script

- **that doesn't use BioPython**
- **that isn't just a Unix/bash script called from Python3...**
- **that you haven't used ChatGPT for...**
- that you will make available as a passworded `ccrypt` encrypted file in a PUBLIC repository on GitHub in your Bxxxxxx-2023 GitHub account

that will allow the user:

1. to identify a family of protein sequences from a user-defined subset of the taxonomic tree (e.g. glucose-6-phosphatase proteins from *Aves* (birds), or ABC transporters in mammals, or kinases in rodents, or adenyl cyclases in vertebrates *etc.*) that could then be processed using, for example, one or more of the EMBOSS programmes installed on the MSc server:
   - to determine, and plot, the level of protein sequence conservation across the species within that taxonomic group
   - to scan the protein sequence(s) of interest with motifs from the PROSITE database, to determine whether any known motifs (domains) are associated with this subset of sequences
   - to do any other appropriate EMBOSS (or other) analysis that you think might add relevant biological information to the outputs
2. write a "help manual" for your programme which has two sections:
   - one aimed at an "ordinary user" (non-coder)
   - one aimed at a competent Python3 code-writer

Tasks in more detail

Write a **generic** (in other words, it will work with anything (sensible) it is asked to do) Python3 programme/script,

- **that doesn't use BioPython**
- **that isn't just a Unix/bash script called from Python3...**
- **that you haven't used ChatGPT for...**
- that you will make available as a passworded `ccrypt` encrypted file in a PUBLIC repository on GitHub in your Bxxxxxx-2023 GitHub account

that will allow **the user** to define a dataset that is of interest to them, which will then be processed to produce the required outputs.

Some or all of these outputs could be generated using EMBOSS programmes already installed. If you want to use an EMBOSS programme, you will need to look at what is available using the EMBOSS link given above, and then chose the most appropriate one(s) to use.

I have suggested a "test set" for you that has a fairly small number of sequences (glucose-6-phosphatase in birds (*Aves*)); please use this "test set" as examples in the help manual.

To summarise, your code will need to be generic:

1. the user of your code will specify the protein family, and the taxonomic group, and then your code will need to obtain the relevant protein sequence data, and perform all subsequent analyses and outputs in the user's space on the MSc server.

   Do **NOT** set it up so it will only work in your workspace! As all files/databases/outputs will be made in the user's homespace on the MSc server, the user's allowable starting sequence set probably shouldn't have more than 1,000 sequences (note, this is just a guideline, not a hard limit!)

   How useful the programme will be *might* depend a bit on how many species are represented in the dataset chosen by the user (i.e. are the sequences all from one species, or are there many different species?), so it would probably make sense to tell the user, and give them the option to continue or not continue with the current dataset?

There are almost certainly (many?) other checks that could/should be done at this stage <u>before</u> continuing to the main processing stages...

2. to determine, and plot, the level of conservation between the protein sequences. Here we are wanting to establish the degree of similarity within the sequence set chosen. The output should go to screen and be saved as a file output.

   Think carefully about how many sequences you might want to use for the conservation analysis. We used a programme that does this sort of thing in the lectures, but this time, should you limit the number of sequences used for the conservation analysis and plotting to some number? If you think you should, working out which ones to keep could be done in several different ways, some a lot easier than others: the method of selection choice is up to you, should you go down this route.
3. to scan protein sequence(s) of interest with motifs from the PROSITE database, to determine whether any known motifs (domains) are associated with this subset of sequences: were there any, and if so, what were their names?
4. a "wildcard" option for you: to do <u>any</u> other appropriate EMBOSS (or other) analysis that you think might add <u>relevant</u> biological information to the outputs
   I am leaving it up to you to chose what might constitute "relevant biolgical information" that your programme will provide to the user: you are training to be a **bio**informatician, so you should be able to decide, and justify your choice!
5. maintain and provide for assessment a full git log history of your code writing/commit activities for this ICA

Some of the programs you might (or might not) need are, or should be, already installed:

- `esearch`, `efetch`, and others for searching and retrieving from any of the NCBI databases from the `edirect` package
- `clustalo` for clustering sequences etc
- `makeblastdb`, `blastn`, `blastx`, `blastp` for doing BLAST analyses etc
- `plotcon` and many others from EMBOSS ; use `-help -verbose` to get more info, or check out the link given above

I have also provided the `pullseq` programme, and have put this as an executable in the `/localdisk/data/BPSM/ICA2/` directory; see https://github.com/bcthomas/pullseq for more details about what it does, so you can then decide whether you will need it or not.

If you need any help figuring out what parameters to use with any of these programmes, usually this can be obtained by typing the programme name followed by `-help` or `--help`. Don't forget, Google is (almost always) your friend!

When putting the programme together, you should consider the following:

- There can be quite a number of different ways to structure a solution. Remember, as you did for ICA1, break up the problem into smaller bits, figure those out, then put them all back together. Draw everything out schematically on paper first, so you know where everything should go/what might be needed both in terms of input and output.
- lots of comment lines in the code (please)
- ALWAYS test the code with lots of print statements, you can comment them out later
- ALWAYS test the code with something that you know will work
- the user may be clever, but they can't read your mind: be explicit in telling them what they can and can't do, preferably at the time they are doing it!
- have lots of "error traps" in your code (e.g. when the wrong thing is input, or there is no output, or ...)
- the code is fine, but there is no output on a different test set: what now?!

Write a "help manual" for your programme, which has two main sections;

- a section aimed at an "ordinary user": it contains instructions for running the program, and interpreting the output; it is aimed more at biologists who may not know much about Python3 coding, they just want the outputs!

  The user section of manual **doesn't** need to be huge. Remember that this is for a non-programmer to read, so you don't have to go into details of what variables you're using, *etc*.. It just has to describe briefly how the programme works, how to use it, and, perhaps, how NOT to use it!?

  You shouldn't need more than 5 pages of text, and probably fewer would be fine. Pictures are good here. You should use the outcomes from the test set as example inputs and outputs in this manual.
- a section aimed at a competent Python3 code-writer. This is usually called a "maintenance manual", which explains/shows how the different parts of the programme fit together. Feel free to use words like variable, function,

iteration, error trap etc!

This bit of the help manual should essentially be a description of the programme components and how they link together, **not** a listing of variables/functions/dictionaries/lists etc. Flow diagrams work very well here.

You shouldn't need more than 5 pages of text, and probably fewer would be fine. Pictures are good here too.

Submitting your ICA2 work

Just like we did for ICA1, none of the files should contain or be called by anything that could lead me to your name. This includes passwords, please..!

Please note that **YOU ARE NOT ALLOWED** to use ChatGPT or similar for any part of this ICA.

You will need to submit two things for ICA2, and both items need to be submitted before the deadline! As you know, it is possible to apply for an extension, but you cannot apply for an extension to an ICA after the submission date has passed. If you don't submit until AFTER the deadline, your work will be subject to late penalties.

1. **50% of the marks:** a PUBLIC GitHub repository called ICA2 (e.g. https://github.com/Bxxxxxx-2023/ICA2) containing a passworded `ccrypt` encrypted file called `Bxxxxxx-2023.ICA2.tar.gz.cpt` (please use **YOUR** B number here, not Bxxxxxx!!). The `Bxxxxxx-2023.ICA2.tar.gz.cpt` file should ONLY contain the ICA2 programme/code files(s) for me to run on our server, as well as the full git log of your ICA2 coding project (e.g generate a full git log `all_the_things_I_did_for_ICA2` file).

   Please ensure that you call the remote GitHub repository ICA2

2. **50% of the marks:** a PDF of the help manuals, submitted via Learn. This should be named with your Exam ID in this manner: `Bxxxxxx-2023.ICA2.pdf` (please use **YOUR** B number here instead of Bxxxxxx). Please make sure that you have included the link to your GitHub ICA2 repository and the decryption key for the `Bxxxxxx-2023.ICA2.tar.gz.cpt` file that you have put in your GitHub ICA2 repository.

Marking

Marks will be awarded for :

1. The user "experience"
   - Utility: does your programme present a seemingly useful tool for biologists?
   - Usability: how easy is your programme to use?
   - Interface: does your programme present a consistent interface to the user?
2. The help manual
   - Readability of documentation: are your manual sections easy to read and understand?
   - General description: is it easy for the user (of any level of competence) to understand what the programe does?
3. The code
   - Does it use Python3 in preference to bash/awk?
   - Does it use many of the Python3 features we have learnt?
   - Correct: does the programme produce the correct output when run?
   - Robust to error: does the programme check for valid input and if not good, fail gracefully with a useful error message?
   - Well-structured: is code divided into logical functions/units, where that is possible?
   - Well-documented: do the comments help a programmer to understand/maintain your code?
   - Concise, well formatted: is your code laid out in a clear, consistent way?
4. The git repository
   - Could I get it from GitHub easily?
   - Was all the required code there?
   - Did the history suggest appropriate frequency and timing of usage of the `git commit -m` command?

Important

In the exercises part of these teaching sessions, many of you have shared ideas and code with each other, and that is definitely to be encouraged, because it is a great way to learn!

However, these ICA components **must be all your own work**.

Al's (hopefully) Helpful Hints...!

These are **not** part of the official specification, but might help when thinking about how to tackle the project.

- Some of the tools that you will/might need to complete the exercises we have not talked about much yet. In particular you may need to know about functions and dictionaries (dicts). Don't worry, both will be covered very soon! There are plenty of other bits to work on in the meantime... Think carefully about the data structure(s) that you will use to store things.
- This project is bigger than any of the exercises we have been doing in class, so you will need to think carefully about how to structure your code. Think about which parts of the problem might make good "stand alone" functions. Think about default values for variables.
- It is far more important to make your code clear and readable. Clear code is frequently fast code.
- If you decide to write a function, make sure you know what the inputs (arguments) and output (return value) are going to be, but I'd recommend that you try it with an "ordinary" bit of code first, and lots of `print` statements so that you can keep track of what you think things are doing.
- The programme does **not** actually require you to write a huge amount of code, even if you first think it does!
- The best way to tackle this kind of problem is to break it down into simpler sub-problems, then write the code for that.... you know, my favourite picture...
- Work on one part of your programme at a time. Remember what I have said in class: you make progress much more quickly if you just write a small piece of code and then test it, than if you try to write the whole thing all at once.
- Think about the various ways that a user could break the code (deliberately or otherwise, due to poor typing skills!): **NEVER** underestimate the ability of a user to break your programme...!
- Remember that you will be marked on the code and the manual, so don't make the mistake of spending all your time working on the code, and only treating the user manual as an afterthought: they are both important. Sometimes, a "sketched out" user manual done before the coding begins actually helps to crystallize ideas as to how the coding might best be done!

- If you are having difficulty with this ICA, PLEASE DON'T wait until six minutes before the deadline to ask for help!

Enjoy!
