Biocomputing II – Coursework guidance

Code documentation

The most important things to show are

1. (for the DB end) how you run the program to create and load the database,
2. the agreed API
3. the call tree (how one routine calls anoother) so people can navigate through the code.

Standarized file headers

All code (including HTML) should have a header documenting the function of the code, how to use it (if it's run from the command line), who wrote it, the date, the version number and a change log.

Standardized function headers

Trying to document calls to subroutines outside the API may not be maintainable. Do it in the code and generate a document automatically if you wish.

For example:

```
Input:   STRING      select   SQL Select statement
Input:   STRING      from     SQL From statement
Return:  STRINGARRAY          Array of result tuples
```

All subroutines should contain a docstring that explains how to use the routine and is formatted in a standard way.

There are programs such as doxygen and Pydoc that can generate web pages of documentation if you format parameters in a standardized way:

- https://stackoverflow.com/questions/34331088/how-to-comment-parameters-for-pydoc
- https://www.youtube.com/watch?v=COaSNOrFGQM

Config file

Avoid all hardcoding (absolute URLs, DB contact info, etc) – place all of this in a config file (e.g. config.py or use a text file – in which case you need some code to read the file).

Code layers

The database end needs to...

1. Parse and read the data – you must parse this yourself and not make use of libraries such as BioPython.
2. Store the data into the database
3. Provide an access layer that allows basic information to be extracted from the database without knowing the schema.

The middle layer needs to:

1. Identify where the coding regions are in the genomic DNA
2. Generate the coding DNA sequence from the DNA

3. Align the protein sequence translation with the DNA coding sequence
4. Identify RE sites
5. Provide a list of available REs to the front end
6. Identify whether an RE has restriction sites within the coding region
7. Count codon usage in a gene
8. Calculate codon usage across all coding regions (perhaps use the subroutine developed for counting codon usage in a single gene) - store this somewhere; perhaps back in the database or in a text file.
9. Extract information from the database layer such as the complete gene list or individual gene information. (Some of these may simply be wrappers to database layer code)

The front layer needs to:

1. Take data from the middle layer and format it for presentation.
2. Allow searches to be specified through forms
3. Perform all HTML/CSS formatting, generate tables, etc.
4. Ensure all HTML and CSS is validated! validator.w3.org jigsaw.w3.org/css-validator

Suggested work plan

Week 0 is the day the project is given to you.

- Week 0-1: Everyone – take a look at the data files, their documentation and the project requirements. Use grep or write small scripts to see what happens in the data. Meet to discuss findings including problems identified and what is needed for the requirements.
- Week 1-2: Everyone – consider what needs to be stored in the database to satisfy the front end requirements. Consider any problems in the data and consider suitable database designs. Meet to choose a suitable database design and start to think about APIs. i.e. If you are the front end developer think about what you need from the business layer (you should just be formatting data and sending off queries). If you are the middle layer developer, work out what you are going to need from the database - you will not write any SQL or any HTML.
- Week 2-3: Discuss and fix the specification for your APIs with the understanding it *may* change later – but you wish to avoid this... Immediately implement dummy versions of all the APIs which return some random piece of data (of roughly the right form) rather than the correct data. At this stage the front end should be able to call the middle layer and get something that could be displayed. e.g. if the front end wants a list of gene IDs with protein names etc., then a set of text should be returned by the middle layer in the agreed data structures. This could even be picked up from random data provided by the database wrapper API.

At this stage you should have an agreed database design with a 'wrapper layer' encapsulating the SQL and providing an agreed API with dummy routines that can be used by the middle layer. Similarly your middle layer provides an agreed API encoded in dummy routines that the front end can use.

Each team member should then be able to work largely independently. There should be no need for any last minute integration of the different layers – the database and middle layer developers should already have provided dummy code which they will replace with the actual working code. As the code gets updated, the functionality will appear.

The team will clearly need regular meetings to discuss any needed changes to the APIs (should be minimal if you have spent enough time discussing it at the start!) and will need to come together in the final couple of weeks to sort out any remaining issues.

Reflective Essay (2-5 pages)

1. Explain how you set about the project detailing:
    - how you interacted with the other members of the group
    - how you (as a group) came up with the requirements for the overall project
    - how you (as an individual) came up with the requirements for your contribution
2. How well did the development cycle work within the group?
3. You should document the development process - what stages did you go through and what steps did you take? Did you go through several iterations or did you spend a long time in design and then a relatively short development stage?
4. What strategies (if any) did you take for testing your code - particularly if other people's code wasn't ready?
5. Are there any known issues or bugs, or anything that doesn't work as specified?
6. You should discuss what worked well and what could have worked better - both with your code and the group interaction. Were there any particular problems? Conversely were there any solutions or ideas that you were particularly proud of?
7. Were there alternative strategies you could have used? (Either for the project design or implementation)
8. What do you think you gained out of the project? How has this experience helped you? What experience or insights have you gained?

Consequently I suggest headings of the form:

1. Approach to the project
    - Interaction with the team
    - Overall project requirements
    - Requirements for my contribution
2. Performance of the development cycle
3. The development process
4. Code testing
5. Known issues
6. What worked and what didn't - problems and solutions
7. Alternative strategies
8. Personal insights

Code documentation (1-2 pages per tier)

Note that this should not simply be a re-printing of the code with the comments in bold (or something similar).

In general:

- Provide a description of the function of subroutines and programs. It should describe WHAT the functions do not HOW they do it (such comments should appear in the code).
- Provide an explanation of the structure of the code - i.e. what calls what - decribing (perhaps as a diagram) how separate pieces of code or subroutines (and HTML for the front end) interact with one another.
- Explain how to install and run the software. For example, how are the programs run to populate the database? Are there other programs that must be run to calculate and store the information on chromosome codon usage before these are stored in the database? If so, how are these run?

For specific layers, the documentation should:

- Front-end provide a short end-user document, and explain what middle layer code was

accessed and how the results were displayed in CGI scripts. You should document any semantic markup used.
- Business logic define the API provided to the front end in such a way that another programmer could implement a front end with reference to your document.
- Data access routines (which may be provided by the database programmer or the business logic programmer) define the API provided to the business logic in such a way that another programmer could implement the business logic with reference to your document.
- Database provide table definitions and UML diagrams (or equivalent) detailing indexes, primary and foreign keys, and any constraints.

Marking criteria

In general we will be taking a loose, "holistic" approach to marking so we are able to give credit where it has been earned rather than penalizing people for not doing enough in one area. You are expected to stretch yourselves!

However, criteria will include:

- How well does the web server meet the required brief?
- How easy is it to use?
- Is the 3-layer design properly implemented?
- Quality of the HTML
    - is it compatible with XHTML (or later) standards?
    - is all formatting done with CSS?
    - does it make appropriate use of semantic markup?
- Quality of code
    - is code properly laid out?
    - is it readable and understandable using suitable variable names?
    - does it use appropriate error and code checking?
    - is it properly commented? This should include a header at the top of each file and a docstring comment for each function. Comments must include author information.
    - are things like hard-coded pathnames, database details, etc appropriately placed to make installation on a different machine straightforward? At a minimum hard-coded information should be at the top of a file - ideally in a separate configuration module or in a file that is read by the code.
- Quality of documentation
    - is all code appropriately documented?
    - is the documentation clear and easy to understand?
    - is the interface well documented - either with a separate user manual or with appropriate help information on the web pages?
- Commentary/Reflection - are the following items addressed
    - How well did the development cycle within the group work?
    - What were the challenges?
    - What might they have done differently?
    - What have you gained from this experience in terms of becoming a better programmer?