

Contact tracing

COMP0015 Term 2 Coursework – 60% of the module

This document explains the arrangements for the coursework. You will create an application that analyses a small dataset to determine how a virus is spreading based on contact tracing data. This document is fairly lengthy; do not be deterred by this. The coursework has been carefully designed so that you can complete it part by part and know that you have the correct functionality at each point. Each part is described in its own section. Start tackling your coursework as early as possible; give yourself time to resolve the issues that you encounter.

Deadline

27 March 2023 at 16:00 (UK time).

How to submit your work

Submit your .py file only at the assignment link on Moodle. Do not submit the data files, we have those. Do not upload a folder containing your files because this can cause compatibility issues for the marking team. You must ensure that your program works properly on your own computer before you submit the code. Make sure your student number (not your name) is included in comments at the top of your program.

Testing

You are responsible for testing your program carefully. Make sure that you have thought about all the things that can go wrong and test your program to ensure that you know it works correctly in all circumstances.

However, as an aid, we have developed a web-based testing service. More details can be found in Appendix 1.

Assessment

You are expected to show that you can code competently using the programming concepts covered so far in the course including (but not limited to): use of files, strings, lists, dictionaries, sets, conditions, loops and functions.

Marking criteria will include:

- Correctness – your code must perform as specified
- You must apply Python concepts appropriately.
- Programming style – see section ‘Appendix 2 Style Guide’ for more detail.
- Your assignment will be marked using the rubric in Appendix 3. This is the standard rubric used in the Department of Computer Science. Marks for your project work will be awarded for the capabilities (i.e. functional requirements) your system achieves, and the quality of the code. Categories 5 and 6 of the rubric will be used for coding assignments.

Plagiarism

Your work will be checked for plagiarism using a world-class plagiarism detection tool, [MOSS](#). MOSS is used in the Department of Computer Science and across UCL more widely. According to UCL policy, plagiarism is defined as the presentation of another person's thoughts or words or artefacts or software as though they were your own. Plagiarism includes copying work from other students, submitting work completed by students in previous years of the course, and copying from journal articles, books and internet sources without correct referencing. Plagiarism seriously undermines the integrity of the College and its graduates and if a deliberate case of plagiarism is suspected in this course it will be treated as cheating under the University of London Proceedings in Respect of Examination

Irregularities. Further details of the policy and proceedings can be found on the College website at: <https://www.ucl.ac.uk/academic-manual/chapters/chapter-6-student-casework-framework/section-9-student-academic-misconduct-procedure>. It is most important that if you feel that you are not able to deal with the study requirements in this course or if you are unsure about referencing conventions, then please ask your lecturer for help. Do not feel tempted to risk your personal reputation and progress through your degree program by plagiarising or cheating.

It is also most important to remember that each assessment task is an opportunity for you to learn and to develop skills that will be of great value in professional and other areas of your life. While you may feel under pressure to complete each assessment task you should not waste important learning opportunities by dishonestly fulfilling the assessment requirements, including copying material directly from the internet. If you are having difficulty meeting assessment deadlines and requirements please contact your Tutor to work out how best to maximise your learning, rather than resorting to plagiarism or cheating.

If you are in any way unsure about the rules and interpretations relating to plagiarism, please contact your personal tutor or the module leader for clarification. Plagiarism will not be tolerated in this module.

Avoiding Plagiarism

Here are some tips to avoid plagiarism.

1. Cite all sources of code that are not your original work. You can put the citation into the comments in your program. For example, if you find and use code from a website:

```
# The following code is from  
# https://www.quackit.com/python/tutorial/python\_hello\_world.cfm
```

2. Citing sources avoids accusations of plagiarism and penalties for academic misconduct. Please note that you may still receive a low grade if you submit code that is not primarily your own. Cited material should never be used to complete assignments.
3. Collaborative coding is strictly prohibited. Your assignment submission must be strictly your own code. Discussing anything beyond assignment requirements and ideas is a strictly forbidden form of collaboration. This includes sharing code, discussing code itself, or modelling code after another student's algorithm. You must not use (even with citation) another student's code.
4. Making your code available, even passively, for others to copy, is also plagiarism.

Before starting your assignment

You are provided with some starter code in file `template.py`. Your first action should be to make a copy of this code into a new file, `contact.py`, where you will work afterwards. You have also been given some text files containing data which you must download and save before starting the assignment. You are also provided with some additional files showing the output that your program should produce for each of the datasets given.

Running the contact tracing program

There are two ways to run the program from the terminal depending on whether you want to provide the data file name on the command line or whether you want the user to be prompted for the file name. The code in `main()` contains code to handle this. Important: you should not edit the code in `main()`.

Entering a file name on the command line

On Windows, run the program in the terminal, specifying the data file name:

```
python contact.py DataSet0.txt
```

or on macos type:

```
python3 contact.py DataSet0.txt
```

The meaning of the terms on this line is:

python or python3	The python interpreter. On macos this will be python3 and on Windows, this will be py or python.
contact.py	The name of the python program.
DataSet0.txt	Name of the data file.

Table 1 Running the program, specifying the file name on the command line

Prompting the user for a file name

To prompt the user for a file name, simply run the program in your editor (IDE) as you would normally.

Section 1: Get started

In this coursework, you must not import any additional libraries apart from those already imported in the starter code.

You have been given several files containing fictitious contact tracing data. This contact tracing data is one-directional, meaning sick people who have tested positive for a fictitious zombie disease have reported the people with whom they have had contact. We don't have data for the reverse direction where a well person had contact with a sick individual. In order to work with the contact tracing data, you will need to load a data file storing contact tracing data and create appropriate data structures such as a dictionary or a list. These data structures can be used to identify the relationship between the individuals in the data.

Many of the sections require you to print out data after you have calculated it. Your printing commands should go into the functions called `pretty_print_section_X()` (where X is a section number).

Important: ensure that your program's output matches exactly the output given to you unless otherwise specified.

Important: do not import any additional packages; lines 7-9 of the `template.py` file already imports `sys`, `os.path`, and `format_list`; that's all you need (and all you are allowed).

Important: do not modify the `main()` function; all of your work should be in the other functions that are labelled for the various sections.

Data files

You may assume that the data in the files you have been given is correct. The data is stored in CSV (comma separated value) files. The files (as well as your assignment's expected output for each file) are posted on Moodle with the assignment information. Each file ends with the .txt extension. You can open the files in Visual Studio Code or in a text editor to see the contents.

Each line in the file will be of the form `<SickPerson1>,<Contact1>,<Contact2>,...,<Contactn>`

`<SickPerson1>,<Contactn>` are placeholders for specific people.

Each line will always begin with exactly one person, followed by one or more other people. For example, lines in the file could be **Jonathan, Alice, Bob** or **Carol, Alice**

The names may include a mixture of upper and lowercase letters and may also include spaces. The end of a name is indicated by comma separator. In order to make the files easier to work with you can assume that there will not be any spaces immediately before or after any of the commas (only within the start and end of a name).

Section 2: Check the file exists (1 mark)

Take a look at the following code in `main()`:

```
# 1. Check that the file exists
if not file_exists(filename):
    print("File does not exist, ending program.")
    sys.exit()
```

The function `file_exists()` takes the file name given as a parameter, and checks that the file exists. Your first task is to complete function `file_exists()`. The function `file_exists()` must return `True` if the file exists and `False` if it does not. Hint: use the function `isfile()` from the python library [os.path](#).

Section 3: Create a dictionary (3 marks for creating the dictionary + 1 mark for correctly printing the error message when needed)

Complete the function `parse_file()`. This function takes a file name as a parameter, reads the file line by line and creates a dictionary with the name of the sick person as the key and a list of contacts as the value. The function will return the dictionary. If a `ValueError` exception is generated when processing a line of the file, the program should print the message:

Error found in file, continuing.

Important: for this section, and all sections that follow, you should print to the screen (rather than, e.g., writing your results to a file). Also, it is critical that you produce the exact error message we specify to get the mark.

Your program should continue if there is an error of this type and attempt to read the next line of the file. If there are no valid lines in the file then the function should return an empty dictionary. Ensure that the file is closed.

Section 4: Print contact records (3 marks)

Please fill in the function body of the function `pretty_print_section_4(contacts_dic)`. Print the names of the sick people with the list of people that they had contact with. For example, if the data file contains the line:

Jonathan, Alice, Bob, Charles

Your output should be:

Jonathan had contact with Alice, Bob and Charles

Important: check the format of the output expected for the files given to you; unless we specify otherwise we expect you to follow this format exactly. The output expected for each data file can be found in the corresponding file name with the word “out” in the name. For example: the output for DataSet1.txt is given in DataSet1-out.txt.

Note that commas appear after all the contacts except the last two items where the word “and” must be used. You have been provided with module `format_list.py` which contains the function `format_list()`. Use the function `format_list()` to print the list as shown in the output.

For example, the output expected for DataSet2.txt is:

Contact Records:

```
Alice had contact with Bob, Carol, Darryl, Ettienne Fourth, Forbert Findlesworth II and Gordie
Bob had contact with Lammle
Carol had contact with Lammle
Darryl had contact with Lammle
Ettienne Fourth had contact with Lammle
Forbert Findlesworth II had contact with Job and Molly
Gordie had contact with Kirk
Hanes had contact with Bob, Carol, Ettienne Fourth and Forbert Findlesworth II
Illersley had contact with Darryl, Ettienne Fourth, Forbert Findlesworth II, Gordie, Job and
Kirk
Job had contact with Lammle
Kirk had contact with Job
Molly had contact with Job and Kirk
```

Important: Notice that within each contract record, the names are sorted (within Alice, Bob ... Gordie); and moreover, the contract records are also sorted (Alice ... Molly). Reminder: if you have a list `mylst`, then `mylst.sort()` will sort it. (There are other ways to sort it too.)

Section 5: Identify possible patients zero (4 marks for correctness + 1 mark for pretty-printing)

The input files consist of contact tracing records where we know everyone tested positive before constructing their contact record list, then we can track back the path of the likely infection to what we will call **patient zero(s)**. One way to think about patient zero(s) is that they are people that are sick who do not appear in anyone else’s contact list.

Your program should list the patient zero(s) for the input data.

For example, the output for DataSet1.txt should be:

Patient Zero(s): Bob, Farley and Larry

Important: Notice that the list is sorted.

Complete function `find_patients_zero()`. Do not change the function signature, this means that the parameters and the return type of function `find_patients_zero()` must remain the same. Complete the function `pretty_print_section_5()` to print the results. Use the function `format_list()` to print the list as shown in the sample output.

Section 6: Identify potential zombies (4 marks for correctness + 1 mark for pretty-printing)

A **potential zombie** is any person that might be infected (because they have been in contact with a sick individual) but who has not yet been conclusively determined to be sick. Remember that a sick person occurs as a first entry in each contact record in the data file so a **potential zombie** will occur in the list of a sick person, but not occur as a sick person themselves. The output from your program should display all of the potential zombies with an appropriate heading. There should be no duplicates in this list.

The output for DataSet1.txt should be:

Potential Zombies: Philip and Sarai

Important: Notice that the list is sorted.

Complete function `find_potential_zombies()`. Do not change the function signature, this means that the parameters and the return type of function `find_potential_zombies()` must remain the same. Add code to `pretty_print_section_6()` to print the results. Use the function `format_list()` to print the list as shown in the output.

Section 7: Identify people who are neither patients zero(s) nor potential zombies (4 marks for correctness + 1 mark for pretty-printing)

The people that are neither patient zero(s) nor potential zombies are all of the individuals that occur in the data file but were not printed in **Section 5: Identify possible patients zero (4 marks for correctness + 1 mark for pretty-printing)** or **Section 6: Identify potential zombies (4 marks for correctness + 1 mark for pretty-printing)**. You must complete function `find_not_zombie_nor_zero()`. You can use the lists returned by the functions created for the previous two sections as arguments to function `find_not_zombie_nor_zero()`. The function should construct a list of names that only includes individuals who were not **potential zombies** and were not **patient zero(s)** and return this list. Do not change the function signature of `find_not_zombie_nor_zero()`. This means that the parameters and the return type of function `find_not_zombie_nor_zero()` must remain the same.

Add code to `pretty_print_section_7()` to print the results. Use the function `format_list()` to print the list. The output for DataSet1.txt should be:

Neither Patient Zero or Potential Zombie: Carol, Leanne, Mark, Paul, Will and Zach

Important: Notice that the list is sorted.

Section 8: Identify the most viral people (4 marks for correctness + 1 mark for pretty-printing)

The most viral people are the people who likely infected the greatest number of other people in the data set because they had the longest list of contacts. You must identify the most viral people by completing function `find_most_viral()`. Do not change the function signature, this means that the parameters and the return type of function `find_most_viral()` must remain the same. Add code to `pretty_print_section_8()` to print the results. Use the function `format_list()` to print the list as shown in the output.

The output for DataSet1.txt should be:

Most Viral People: Bob

Important: Notice that the list is sorted.

DataSet1.txt has only one most viral person but other data sets may have several most viral individuals, all of whom infected the same amount of people. When that situation occurs, your program should display all of the most viral people.

Section 9: Identify the most contacted person (4 marks for correctness + 1 mark for pretty-printing)

The most contacted person is the member of the data set who has possibly been infected by (been in contact with) the most sick members of the data set. You must identify the most viral people by completing function `find_most_contacted()`. Do not change the function signature, this means that the parameters and the return type of function `find_most_contacted()` must remain the same. Remember, there should be no duplicates in this list returned from the function. Add code to `pretty_print_section_9()` to print the results. Use the function `format_list()` to print the list as shown in the output.

The output for Part 9 for DataSet1.txt should be:

Most contacted: Leanne and Mark

Important: Notice that the list is sorted.

Section 10: Find the maximum distance from a potential zombie (4 marks for correctness + 1 mark for pretty-printing)

The maximum distance from a potential zombie is the **longest** contact tracing path downwards in the data set from a sick person to a potential zombie. Using this definition, all potential zombies (people that are not yet confirmed to be sick) have maximum distance 0. Any person that only has contact with potential zombies has maximum distance 1. All other people in the data set have a maximum distance which is one more than the maximum distance of the person they've had contact with who has the largest maximum distance value.

You can determine the maximum distances of all the people in the contact tracing data using the following algorithm:

```
set the max distances of all people, including potential zombies, to 0
set changed to true
while something has changed
    set changed to false
```

```
for each person, p1, in the dataset
    for each person, p2, that p1 had contact with
        if the max dist of p1 <= max dist of p2
            set the max dist of p1 to the max dist of p2 + 1
        set changed to true
```

Implement the algorithm by completing function `find_maximum_distance_from_zombie()`. Do not change the function signature, this means that the parameters and the return type of function `find_maximum_distance_from_zombie()` must remain the same. Add code to `pretty_print_section_10()` to print the results.

The output for DataSet1.txt should be:

```
Heights:
Bob: 4
Larry: 4
Carol: 3
Farley: 3
Will: 3
Mark: 2
Paul: 2
Leanne: 1
Zach: 1
Philip: 0
Sarai: 0
```

Important: Your output may not appear in the same order, this is acceptable; the list need not be sorted.

Sections 11, 12, 13, & 14: Additional credit (2 + 2 + 2 + 4 = 10 marks in total)

So far you have had the opportunity to earn $1 + 4 + 3 + 5 * 6 = 38$ marks. Up to an additional 12 marks can be awarded for good style (including good comments), as discussed further below. This leaves 10 remaining marks for those who like a challenge, of which several are given below.

You should only attempt an additional challenge if you have satisfied all requirements for the coursework, since these marks are more work for less credit. Note: You are strongly encouraged to follow the specification carefully and to use programming techniques as described in the course materials and textbooks. Poor quality code with additional functionality will not improve your marks.

Identify more features in the data (2 marks each, 6 total)

Identify the spreader zombies, regular zombies, and zombie predators. Use the functions `find_spreader_zombies()`, `find_regular_zombies()`, and `find_predator_zombies()` to do so, as well as the associated `pretty_print_section_X()` functions.

Important: you will also need to add docstrings for the `find_X()` functions, as discussed below in the style guide.

A spreader zombie is sick person that only has had contact with potential zombies. Display the spreader zombies under an appropriate heading. If there aren't any spreader zombies then you should display the heading, followed by "(None)". In `DataSet1.txt`, you will see that Leanne has had contact with Sarai and Zach has had contact with Philip. Both Sarai and Philip are both potential zombies and this means that Leanne and Zach are spreader zombies.

A regular zombie is a sick person that has had contact with both potential zombies and people who are already sick. Display the regular zombies under an appropriate heading. If there aren't any regular zombies then you should display the heading, followed by "(None)". In `DataSet1.txt`, we can see that a regular zombie cannot be a spreader zombie so that excludes Leanne and Zach. Mark has had contact with both Philip and Zach; Philip is a potential zombie and Zach is a sick person so Mark is a regular zombie.

A zombie predator is a person that only has contact with people who are sick. Display the zombie predators under an appropriate heading. If there aren't any zombie predators then you should display the heading, followed by "(None)". In `DataSet1.txt`, zombie predators are Bob, Carol, Farley, Larry, Paul and Will have only had contact with sick people. They have not had contact with potential zombies.

The output for spreader, regular, and predator zombies for `DataSet1.txt` are shown below:

For additional credit:

Spreader Zombies: Leanne and Zach

Regular Zombies: Mark

Zombie Predators: Bob, Carol, Farley, Larry, Paul and Will

Cycle detection (special challenge question, 4 marks)

There is a function called `find_cycles_in_data()` that is called before `find_maximum_distance_from_zombie()`. This is because if you pass a data set with cycles into the latter function, you may end up in an infinite loop!

In the context of our problem a cycle might look something like this:

Tina,
Carol, Tina

Carol

This forms a cycle as Tina infects Carol, and Carol also infects Tina. In this case, the maximum distances will be infinite, which is why section 10 will loop forever.

Accordingly, it is a good idea to check first if your data set contains any cycles. That is why you have been tasked with a final challenge task, to write a function which can detect if there are any cycles in the data set or not.

The `find_cycles_in_data()` function will require you to take as input a contacts list as usual and return either true or false as to whether that data set has a cycle or not. You may use the web to learn a bit about cycles in (directed) graphs (e.g. https://en.wikipedia.org/wiki/Cycle_%28graph_theory%29).

Two data sets contain cycles to test your solution to this problem against: `DataSetCycle1.txt` and `DataSetCycle2.txt`. Do not use these files until you are ready to try solving this problem.

Good luck and enjoy!

Appendix 1: Testing

We have provided you with a webform on which you can test your coursework submissions to get some feedback on them before you submit your final version for grading. The webform will test how correct your submissions are and will test how it outputs the data to the console (via the pretty printing functions). It will not check for style (12 marks in total), which is graded manually. Thus, if you do perfectly on all 14 sections above, it will be able to award a maximum of 48 marks.

You will receive a final percentage which gives you how correct your code is (as a percentage) and for each section you will receive two percentages, one for functional correctness and the other for the correctness of your output data. The grading of the output is not dependent on the data produced by the corresponding function, i.e. if your main function for a section is incomplete, but you want to test the function which will print the output message for that section, you can do so.

To access this webform you must be connected to the UCL network. If you are not connected to the eduroam network at UCL you may access the webform using the UCL ISD Desktop@UCL Anywhere service (a guide on how to use it can be found here: <https://www.ucl.ac.uk/isd/services/computers/remote-access/desktopucl-anywhere>). Once you are logged on to the UCL network, you can access the webform by using your favourite web browser and searching for the url: <http://comp0015cwtst.cs.ucl.ac.uk>.

Once the webform has loaded you must type in your student number. Student numbers not associated with this course will not be allowed to use this webform so please ensure you type in your 8-digit student number correctly. (If you believe you should be allowed access to this resource but your student number is not working, please contact ylbuzoku@cs.ucl.ac.uk.)


You must then select your Python file by clicking the browse button. This must be a Python file ending with the usual .py Python file extension. Once you have selected your file, you can click the submit query button. Please then wait until the page refreshes. Each submission is given a unique identification number so if something goes wrong whilst grading, or if you believe that the result returned to you by the tool is not accurate, you can email the above quoted email with your submission number. However, it is much more likely that your code has a bug rather than ours, so please first read the error that is returned by the webform before emailing as it might provide you with some hints as to what might be going wrong with your code.

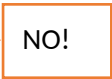
Note that if a function takes longer than 30 seconds to complete, the webform will declare that function as being “stuck” and will mark that function with a 0. Furthermore, you may only make one submission every 90 seconds.

Appendix 2: Style Guide (12 marks in total)

You must adhere to the style guidelines in this section.

1. Write useful comments, as and when appropriate, so that your code is clear to a human.
2. Use Python style conventions for your variable names (snake case: lowercase letters with words separated by underscores (_) to improve readability).
3. Choose good names for your variables. For example, `num_bright_spots` is more helpful and readable than `nbs`.
4. Name constants properly using capital letters and put them at the top of your program.
5. The only lines of code in your program that are outside of a function definition are constants, if any, import statements, if any and the call to the main function which is normally in the last lines of the file.
6. Indentation: use a tab width of 4 or 8. Your IDE should do this automatically for you. The best way to make sure your program will be formatted correctly is never to mix spaces and tabs -- use only tabs, or only spaces.
7. Put a blank space before and after every operator. For example, the first line below is good but the second line is not:

`b = 3 > x and 4 - 5 < 32` 

`b= 3>x and 4-5<32` 

8. Each line must be less than **80 characters** long *including tabs, spaces and comments*. You should break up long lines using `\`. You don't need a continuation character for comments or if you are breaking up the parameters of a function.
9. Function names should also be in `snake_case`: `encrypt_message()`, `print_introduction()`.
10. Functions should be no longer than about 15 lines in length. Longer functions should be decomposed into 2 or more smaller functions.
11. Variables with global scope should not be used, constants are an exception to this rule. The keyword `global` should not be used.
12. Do not define one function inside another function.
13. Do not use `break` and `continue` inside loops. In general, the use of `break` and `continue` can be avoided by using a combination of `if` statements and writing better conditions on your `while` loops.

Docstrings

Add docstring comments for the extra credit functions (if you do them), and for any other functions you define. Look at the code you have been given for some examples. Your comments should:

1. Describe precisely *what* the function does.
2. Do not reveal *how* the function does it.
3. Make the purpose of every parameter clear.
4. Refer to every parameter by name.
5. Be clear about whether the function returns a value, and if so, what.
6. Explain any conditions that the function assumes are true. Examples: `"n is an int"`, `"n != 0"`, `"the height and width of p are both even."`
7. Be concise and grammatically correct.
8. Write the docstring as a command (e.g., `"Return the first ..."`) rather than a statement (e.g., `"Returns the first ..."`)

Note: you can use any style of docstring you like, your editor may have a default style – you are welcome to use this. Available styles include: NumPy, Google, ReStructuredText.

Here is a template for a NumPy-style docstring

```
def template(arg1, arg2):  
    """  
    Summary line.  
  
    Extended description of function.  
  
    Parameters  
    -----  
    arg1 : int  
        Description of arg1  
    arg2 : str  
        Description of arg2  
  
    Returns  
    -----  
    int  
        Description of return value  
  
    """  
    pass
```

Here is a completed example:

```
def random_number_generator(start_range, end_range):  
    """  
    Return a random number from the range specified. Range includes the start and end of the  
    range.  
  
    Parameters  
    -----  
    start_range : int  
        Start of the range, inclusive  
    end_range: int  
        End of the range, inclusive  
  
    Returns  
    -----  
    int  
        Random number  
  
    """
```

Appendix 3: Computer Science Grading Criteria

Categories 5 and 6 will be used for assessing programs.

		Fail		Pass (2:2)		Merit (2:1)		Distinction (1 st)		
		Inadequate	Weak	Satisfactory		Good		Excellent	Outstanding	Exceptional
		Below 40; BSc: Fail MEng: Fail	40-49; BSc: 3rd MEng: Fail	50-54: Low pass	55-59: High pass	60-64: Low merit	65-69: High merit	70-79	80-89	90+
1-19: Misunderstanding of assignment or similar 20-29: 5 inadequate 30-34: 4 inadequate 34-39: 3 inadequate										
1	Quality of the response to the task set: answer, structure and conclusions	Either no argument or argument presented is inappropriate and irrelevant. Conclusions absent or irrelevant.	An indirect response to the task set, towards a relevant argument and conclusions.	A reasonable response with a limited sense of argument and partial conclusions.		A sound response with a reasonable argument and straightforward conclusions, logical conclusions.		A distinctive response that develops a clear argument and sensible conclusions, with evidence of nuance.		Exceptional response with a convincing, sophisticated argument with precise conclusions.
2	Understanding of relevant issues	Misunderstanding of the issues under discussion.	Rudimentary, intermittent grasp of issues with confusions.	Reasonable grasp of the issues and their broader implications.		Sound understanding of issues, with insights into broader implications.		Thorough grasp of issues; some sophisticated insights.		Exceptional grasp of complexities and significance of issues.
3	Engagement with related work, literature and earlier solutions	Very limited or irrelevant reading.	Significant omissions in reading with weak understanding of literature consulted.	Evidence of relevant reading and some understanding of literature consulted.		Evidence of plentiful relevant reading and sound understanding of literature consulted.		Extensive reading and thorough understanding of literature consulted. Excellent critical analysis of literature.		Expert-level review and innovative synthesis (to a standard of academic publications).
4	Analysis: reflection, discussion, limitations	Erroneous analysis. Misunderstanding of the basic core of the taught materials. No conceptual material.	Analysis relying on the partial reproduction of ideas from taught materials. Some concepts absent or confused.	Reasonable reproduction of ideas from taught materials. Rudimentary definition and use of concepts.		Evidence of student's own analysis. Concepts defined and used systematically/ effectively.		Evidence of innovative analysis. Concepts deftly defined and used with some sense of theoretical context.		Exceptional thought and awareness of relevant issues. Sophisticated sense of conceptual framework in context.
5	Algorithms and/or technical solution	No solution to the given problem, completely incorrect code for the given task.	Rudimentary algorithmic/technical solution, but mostly incomplete.	Reasonable solution, using basic required concepts, several flaws in implementation.		Good solution, skilled use of concepts, mostly correct and only minor faults.		Excellent algorithmic solution, novel and creative approach.		Exceptional solution and advanced algorithm/technical design.
6	Testing of solution (e.g., correctness, performance, evaluation)	No testing or evaluation done.	Few test cases and/or evaluation, but weak execution.	Basic testing done, but important test cases or parts of evaluation missing or incomplete.		Solid testing or evaluation of solution, well done evaluation with good summary of findings.		Very well done test cases, excellent evaluation and very high quality summary of findings.		Exceptionally comprehensive testing, extremely thorough approach to testing and/or evaluation.
7	Presentation or demonstration of solution	presentation or demonstration, very low quality.	presentation or demo of the solution.	present and/or demonstrate solution and summarise work in appropriate format.		demo, persuasive and compelling.		Use of presentation medium with professional style.		presentation, exceptional quality of demonstration.
8	Writing, communication and documentation	Style and word choice seriously interfere with comprehension.	Style and word choice seriously detract from conveying of ideas.	Style and word choice sometimes detract from conveying of ideas.		Style and word choice work well to convey most important ideas. Well documented.		Style and word choice show fluency with ideas and excellent communication skills.		Reads as if professionally copy edited. Exceptional high quality of writing.
9	Formatting aspects, visuals, clarity, references	Poorly formatted, inappropriate visuals, and incorrect reference formatting.	Formatting, visuals and referencing seriously distract from argument.	Formatting, visuals and referencing sometimes distract from argument.		Formatting well-done and consistent, good visuals and consistent referencing.		Formatting, visuals and referencing are impeccable.		Exceptional presentation, impeccable formatting of the document and references.