Northeastern University London

Assessment Brief: Individual Coursework 2024–25

Assessment Details

Course Title:	Fundamentals of Computer Science I
Course Code:	LCSCI4207
Course Leader:	Alexandros Koliousis
Level:	4
First, Second, or Third Sitting:	First
Assessment Title:	Question Time, Part II
Assessment Number:	AE3
Assessment Type:	Group Project
Restrictions on Time/Length:	24 – 32 hours
Assessment Weighting:	30%
Issue Date:	8 November 2024
Hand-in Deadline:	4 December 2024, 13:00
Planned Feedback Deadline:	28 calendar days after hand-in deadline
File Format Accepted:	.kts, .pdf and .txt
Mode of Submission:	Online (Canvas/Gradescope)
Anonymous Submission:	Yes

Assessment Task

The setting

In this project, you will improve upon your Project 1 (AE2) in multiple ways in terms of (**i**) the design of your data types and functions and (**ii**) new functionality added to your application.

Data types and function design

1. You will enhance each *question-answer pair* to support an arbitrary number of "tags" (i.e., string labels).

2. You will generalise the meaning of a *question bank* such as to be agnostic as to the very meaning of questions (and thus support a wider variety of question banks).
3. You will enhance the menu system to be more reusable, as well as support quitting (i.e., leave without forcing a selection).

New application features

4. You will implement methods for interpreting self-reported correctness of a question, this time using *Machine Learning* (ML) and, in particular, *Natural Language Processing* (NLP). The user will be able to select which method to use.
5. When a user will not get a question correct (via self-report), that question will be placed back at the end of the question bank. Thus a bank is only completed when a user gets all questions correct!
6. You will provide question bank options that are a subset of questions that have a particular tag (e.g., all "hard" questions, or those in the topic of "science").
7. Once the program is run, the user will be able to study as many question banks as they wish, selecting subsequent banks from the menu until they quit.

Design overview

You will be designing this program step-by-step. When working on this new and enhanced program, you are welcome to draw upon your Project 1 (AE2) solution or our sample solutions as you see fit and helpful.

Step 1. Questions (8 marks)

Design the data type `TaggedQuestion` to represent a single question-answer pair. You should be able to represent the text for the question and the text for the answer, as well as any number of textual tags (e.g., “hard” or “science”). Tags should not come from any fixed set of options. Creators should rather be able to categorise their questions as they deem appropriate.

A `TaggedQuestion` should have at least two methods:

1.1. `taggedAs` determines if the question has a supplied tag (e.g., has it been tagged as “hard”?).

1.2. `format` produces a textual representation of the question as “$question|answer|tag_1,tag_2,tag_3,...$” (e.g., “`What is the capital of England?|London|easy,UK`”). All three parts are separated with the “pipe” (“`|`”) character; and tags are further separated with a comma (“`,`”).

Include *at least* three examples – they will come in handy later for tests anyway! – and make sure to test the required methods.

Step 2. Files of tagged questions (6 marks)

Now that we have our updated questions, let's update how we read them from files.

2.1. Design the function `stringToTaggedQuestion` that takes a string, assumed to be in the format described in 1.2 above, and produces the corresponding tagged question.

2.2. Design the function `readTaggedQuestionBank` that takes a path to a file and produces a list of tagged flash questions. If the file does not exist, return an empty list. Otherwise, you can assume that every line is formatted correctly (see above).

Step 3. Question bank design (26 marks)

If you think about it, once a question bank has been selected, our application does not need much information about questions to work. In fact, it does not even need the concept of a *question*. The selected bank is either completed, or amidst showing a question, or amidst showing an answer:

```
/**
 * The bank is either completed,
 * showing a question or showing
 * an answer
 */
enum class QuestionBankState { COMPLETED, QUESTIONING, ANSWERING }

/**
 * Basic functionality of any question bank
 */
interface IQuestionBank {

    /**
     * Returns the state of a question bank.
     */
    fun getState(): QuestionBankState

    /**
     * Returns the currently visible text (or null if completed).
     */
    fun getText(): String?

    /**
     * Returns the number of question-answer pairs.
     * (Size does not change when a question is put
     * to the end of the question bank.)
     */
    fun getSize(): Int
```

```
    /**
     * Shifts from question to answer. If not QUESTIONING,
     * returns the same IQuestionBank.
     */
    fun show(): IQuestionBank

    /**
     * Shifts from an answer to the next question (or completion).
     * If the current question was answered correctly, it discards
     * it. Otherwise it cycles the question to the end.
     *
     * If not ANSWERING, returns the same IQuestionBank.
     */
    fun next(correct: Boolean): IQuestionBank
}
```

3.1. Design `ListBasedQuestionBank` to implement the `IQuestionBank` interface for a supplied list of tagged questions. For this problem, your class: (***i***) must have **no** mutable state; and (***ii***) all member data should be private.

3.2. Design `AutoGeneratedQuestionBank` to implement the `IQuestionBank` interface. You are **not** allowed to generate any questions, **nor** have mutable state. The goal is to act as though it had a list of questions automatically produced (see `perfectCubes` in Project 1 (AE2)), but without ever having to generate all those questions. Again, as it is generally good practice, keep all your member data private.

Hint. You will still need to keep track of the **sequence** of upcoming numbers as some of them may get cycled back due to incorrect responses.

Step 4. Menu design (12 marks)

Let's improve the `chooseBank` function from Project 1 (AE2) in two ways:

- In Project 1, your function allowed users to select from amongst a list of question banks. This means you would have to copy-paste your code if you wanted to have a menu of other data. So let's make the function agnostic to the type of the list items being selected.
- In Project 1, the menu did not offer users the possibility of not selecting an option. Let's enable quitting!

Design the function `chooseMenu` that takes a list of any type, as long as it implements the `IMenuOption` interface (see below), and produces a corresponding numbered menu: 1 for the first list item, 2 for the second, and so on; 0 is reserved for quitting the menu (indicating a user's desire to quit without choosing an option). For

each item, the function also shows its *title*. The function returns either the list item corresponding to the number entered, or `null` if 0 was entered. Keep displaying the menu until a valid menu selection (or quitting) is indicated.

```
/**
 * The only required capability for a menu option
 * is to be able to render its title.
 */
interface IMenuOption {
    fun getTitle(): String
}

/**
 * A menu option with a single value and name.
 */
data class NamedMenuOption<T>(
    val option: T,
    val title: String
) : IMenuOption {

    override fun getTitle(): String = title
}
```

Step 5. Sentiment analysis (12 marks)

Consider self-reported correctness. When prompted whether they got a question right, users would type a word starting with “Y” or “y” (most likely, “`Yes`”) to indicate a positivity. In ML, this is an approach to *sentiment analysis*: a problem in NLP that seeks to analyse text to understand its emotional tone (e.g., “`You bet!`” is a positive answer, “`Nope.`” is a negative one). What you have been building is, in a sense, a binary classifier of text, albeit a naive version of it (likely, “`You bet!`” also starts with “Y”). Your goal is now to try and use a more sophisticated approach to sentiment analysis: one that learns to distinguish positive or negative sentiment in text based upon a data set of *labelled examples*:

```
data class LabeledExample<E, L>(val example: E, val label: L)
```

The above data class associates a *label* (e.g., a `Boolean`, an `Int`, or a `String`) with an *example* (e.g., a `String`, a `List<Int>`. Here is one such dataset:

```
val dataset: List<LabeledExample<String, Boolean>> =
    listOf(
        /* Some positive examples */
        LabeledExample("yes", true),
        LabeledExample("y", true),
        LabeledExample("indeed", true),
```

```
        LabeledExample("aye", true),
        LabeledExample("oh yes", true),
        LabeledExample("affirmative", true),
        LabeledExample("roger", true),
        LabeledExample("uh huh", true),
        LabeledExample("true", true),
        /* Some negative examples */
        LabeledExample("no", false),
        LabeledExample("n", false),
        LabeledExample("nope", false),
        LabeledExample("negative", false),
        LabeledExample("nay", false),
        LabeledExample("negatory", false),
        LabeledExample("uh uh", false),
        LabeledExample("absolutely not", false),
        LabeledExample("false", false),
    )
```

Note. We call this data set balanced since it has an equal number of examples for each label. Balanced data sets are one tool (of many) when trying to avoid algorithmic bias!

Notice that our simple heuristic based on the first letter of a `String` is pretty good according to this dataset, but will make some lucky guesses (e.g., "`false`") and some actual mistakes (e.g., "`true`"). One approach we **could** take is just to have the computer learn by rote memorisation. That is, respond with the labelled answer from the dataset. But what about if the student supplies an input not in this list above? The approach we will try as a way to handle this situation is the following:

- If the response is known in the dataset (independent of uppercase or lowercase), use the associated label.
- Otherwise, find the 3 "closest" examples and respond with a majority vote of their associated labels.

This algorithm will represent our attempt to *generalise* from the data set. We know we will always get certain responses correct, and we will let our data set inform the response of unknown inputs. As with all approaches based on ML, this approach is likely to make mistakes – even those that we'll find confusing or even comical! – and we should be judicious in how we apply the system in the world. Now let's build up this new classifier, step-by-step!

5.1. When finding the “closest examples”, it will be helpful to be able to get the **top-k** of a list by some measure (e.g., a function that can get the top-3 strings in a list by length, the top-1 (i.e., best) song by ratings, and so on).

Design the function `topK` that takes as arguments: (***i***) a list of items, (***ii***) a corresponding metric function, and (***iii***) the number k (assumed to be a positive integer); and returns the k items in the list that get the highest score (in case of ties, you are free to return any of the winners).

5.2. What does it mean for two strings to be "close"? There are actually multiple reasonable ways of capturing such a distance, one of which is the *Levenshtein Distance* (see Wikipedia). The *Levenshtein Distance* describes the minimum number of single-character changes (e.g., adding a character, removing one, or substituting) required to change one string into another.

Design the function `levenshteinDistance` that computes this distance for two supplied strings. If you look at the Wikipedia article, the Definition section is really what you want to translate to Kotlin.

Note. This is a common task in computing – translating theoretical and mathematical description of an approach into code for your system. You have also been supplied some tests to help make sure you got it right. Be sure to make sure you understand the metric and tests before you start coding!

5.3. Implement a *k-Nearest Neighbour* (k-NN) classifier.[1] Given a query, a dataset of labelled examples, a distance function, and a number k, let the k closest elements of the dataset "vote" (with their label) as to what the label of the query should be. Design the `getLabel` function:

```
typealias DistanceFunction<T> = (T, T) -> Int
```

Since this method might give an incorrect response, we will return not only the predicted label, but the number of "votes" received for that label (out of k):

```
data class ResultWithVotes<L>(val label: L, val votes: Int)

/**
 * Uses k-NN to predict the label for a supplied query
 * given a labelled data set and a distance function.
 */
fun <E, L> nnLabel(
    query:   E,
    dataset: List<LabeledExample<E, L>>,
    distFn:  DistanceFunction<E>,
    k:       Int,
): ResultWithVotes<L> {
    /* See starter code for help. */
}
```

[1] You can read online descriptions, e.g. on Wikipedia, for lots of details and variants of k-NN, but we will give you all the information you need here.

5.4. Now we can put some pieces together. Design the function `classifier` that checks to see if a query exists in the dataset; and, if not, performs a 3-NN classification using the data set and Levenstein distance metric.

Step 6. Putting all together (16 marks)

Now let's put it together and study!

6.1. Design the function `studyQuestionBank` that uses the `reactConsole` function from the Khoury library to study a supplied question bank and using a supplied classifier to interpret self-reported correctness. The function should return how many questions were in the question bank and how many total attempts were required to correctly respond to all of them:

```
data class StudyQuestionBankResult(
    val questions: Int,
    val attempts:  Int
)
```

In case it helps, here is a trace of a short study session:

```
What is the capital of Massachusetts, USA?

_
Boston
Correct ((y)es/(n)o)?
yup
What is the capital of California, USA?

_
Sacramento
Correct ((y)es/(n)o)?
no :(
What is the capital of the United Kingdom?

_
London
Correct ((y)es/(n)o)?
YES!
What is the capital of California, USA?

_
Sacramento
Correct ((y)es/(n)o)?
```

```
yessir!

Bye.Questions: 3, Attempts: 4
```

6.2. Finally, design the program `study` that:

6.2.a. Uses `chooseMenu` to select from amongst a list of question banks. The options must include at least one question bank read from a file using `readTaggedQuestionBank`; one `AutoGeneratedQuestionBank`; and one that filters based upon a tag being present (e.g., only "hard" questions from a list). The latter can be the questions read from a file.

6.2.b. Uses `chooseMenu` to select one of two sentiment analysis functions: our naive classifier (based on whether the answer starts with "Y" or "y"); or our more advanced one (see Step 5).

6.2.c. Uses `studyQuestionBank` to study through the selected question bank with the selected sentiment analysis function.

6.2.d. Returns to 6.2.a and continues until either of the two menus indicate a user's desire to quit.

Step 7. Individual report (20 marks)

Write an **individual report (up to 2 pages, 1' margins, 11pt font)** reflecting on your contributions to the project: (***i***) What did you do for the project? (***ii***) What were the challenges you faced? And (***iii***), how did you overcome them?

Assessment Criteria

You will be evaluated on several criteria, including: (***i***) adherence to instructions and the Fundies 1 coding style guide; (***ii***) correctly producing the functionality of the program; (***iii***) design decisions that include choice of tests; (***iv***) appropriate application of coding abstractions; (***v***) task- and type-driven decomposition of functions; and (***vi***) your individual report.

70 or higher

There was evidence of the ability to perform all programming tasks correctly. The demonstration of the methods was excellent, coherent, well documented and clearly explained.

60-69

There was evidence of ability to perform some programming tasks correctly. The demonstration of the methods is good, coherent and reasonably detailed and explained.

50-59

There was evidence of ability to perform some programming tasks correctly, but the demonstration of the methods was limited, incoherent, not adequately documented and vaguely explained.

40-49

There was limited evidence of ability to perform programming tasks. The demonstration of the methods involved significant omissions and produced substantial inaccuracies.

39 or less

Failure to solve the programming task in assignment. Methods were completely incorrect or absent. General grading criteria for Level 4 are described in Appendix B of the course syllabus.

Submitting Assessments

1. Your submission should be **anonymous**. Remove anything from your code that can identify you before submission.
2. You are to upload a file `question-time-part-ii.main.kts` file, a `questions.txt` file, and a `report.pdf` file on Canvas/Gradescope for your submission:
    2.1. Your code should be **the same** as the code submitted by any other member of your group.
    2.2. Your text file should contain the question-answer pairs (and their tags) read with the `readTaggedQuestionBank` function.
    2.3. Your report should be an **up to 2-page individual reflection** on your contributions to the project:

    What did you do for the project? What were the challenges you faced, and how did you overcome them?
3. You are expected to provide necessary documentation for your code. Part of your marks will be allocated on the quality of your comments!
4. Since mutation has not been covered extensively in class, your program is **not allowed** to make use of mutable variables, including mutable lists.
5. All interactive parts of your program **must** make effective use of the `reactConsole` function.
6. Staying consistent with our style guide. All functions must have a preceding comment specifying their purpose and an associated `@EnabledTest` function with sufficient tests using `testSame`.

7. All data must have a preceding comment specifying what it represents and associated representative examples.

You have three submission attempts, but only the last submission will be graded. If your last submission attempt is late, you will receive the late penalty even if you have a previous submission that was on time. Please make sure to avoid multiple submissions for assessments with multiple components, as only the last attempt will be graded. Upload several files in one submission attempt instead.

If your assessment requires anonymous submission (see the assessment details table at the top of your assessment brief), please be sure you have left your name off of your submission and out of the submission file name, as failing to do so may result in a 0% mark on the assessment.

Refer to the assessment details table in your assignment brief for acceptable file formats. Avoid submitting .zip files (unless explicitly required by the assessment brief); use the 'add files' function to submit multiple files instead. If you are submitting a physical artefact, you must also provide clear and thorough documentation (such as in the form of photographs or a video) of your submission by the deadline; see the bottom of this section for guidance on submitting video files.

Please ensure that you tick the agreement box at the very bottom of your Canvas submission page (scroll down if you don't see it). This will enable you to select 'Submit Assessment.' Please review the submitted file to ensure that everything is in order.

If you encounter any issues with submission, e-mail a copy of your assignment before the deadline to student.assessments@nulondon.ac.uk along with screenshots of the problem on Canvas, showing a timestamp.

To turn on notifications for submission confirmation emails in your Canvas settings: Account > Notifications > Turn on the bell for 'All submissions.' In the app this is via Settings > Email Notifications > All submissions.

To submit a video recording: Select the 'Panopto video' icon in the text entry box in your submission portal. You can upload a video file of any format from your media library by selecting 'upload,' choosing 'my folder' in the drop down menu, and clicking 'insert.' You should be able to play the video back once it processes. See further explanation, including guidance on recording videos using Panopto, in this support article: 'How to Submit a Video Assignment in Canvas.'

Marking

The University uses two categorical assessment marking schemes – one for undergraduate and one for postgraduate – to mark all taught programmes leading to an award of the University.

More detailed information on the categorical assessment marking scheme and the criteria can be found in the Course Syllabus, available on the University's VLE.

Learning Outcomes

This assessment will enable students to demonstrate in full or in part the learning outcomes identified in the Course Descriptor.

On successful completion of this assessment, students should be able to:

Knowledge and Understanding

K1a Demonstrate knowledge and understanding of the underlying principles and concepts of programming.

K2a Demonstrate knowledge and understanding of program design principles and concepts such as parametric polymorphism (e.g., generic functions and data types), generative recursion, and accumulators.

K3a Demonstrate an understanding of the technical, social and management dimensions of programs, their extensibility and correctness, in real-world applications.

Subject-Specific Skills

S1a Complete a data analysis of a problem statement and describe the data required to solve a problem; create input and output examples of the data to describe the desired functionality of a program that solves the problem at hand; and use aforementioned examples for testing.

S2a Plan an iterative approach to solve large problems; and design scalable, abstract data collections to solve growing problems.

S3a Compose programs from several functions and data collections, either sequentially (e.g., for batch applications) or using event-based features (e.g., for web applications).

Transferable Skills

T1a Make reasoned discussions and contributions in group settings, fostered through collaborative work during lab sessions.

T3a	Display a developing technical proficiency in written English and an ability to communicate clearly and accurately in structured and coherent pieces of writing.

Accessing Feedback

Students can expect to receive feedback on all summative coursework within 28 calendar days of the submission deadline or, if applicable, the last oral assessment date, whichever later. The 28 calendar day deadline does not apply to work submitted late. Feedback can be accessed through the assessment link on the Canvas course page.

Late Submissions

Please ensure that you submit your assignment well before the deadline to avoid any late penalties, as a submission made exactly on the deadline will be considered late. Please keep in mind that there may be differences between your computer's clock and the server time, which can cause discrepancies, and that Canvas may take some time to process your submission.

Your Canvas submission portal displays two due dates: one is the deadline for your assignment, and the second is the latest possible date by which your assignment can be submitted late. Please make sure you submit by the assessment deadline in order to avoid late penalties.

If assessments are submitted late without approved Extenuating Circumstances, there are penalties:

- For assessment elements submitted up to one day late, any passing mark will receive 10 marks deducted or a threshold pass (40% for undergraduate students, 50% for postgraduate students), whichever is higher. Any mark below 40% for undergraduate students and below 50% for postgraduate students will stand.
- Students who do not submit their assessment within one day of the deadline, and have no approved Extenuating Circumstances, are deemed not to have submitted and to have failed that assessment element. The mark recorded will be 0%.
- For assessment subelements, late submission will result in non-submission penalties deducted according to the marking criteria above.

For further information, please refer to AQF7 Part C in the Academic Handbook.

Extenuating Circumstances

The University's Extenuating Circumstances (ECs) procedure is in place if there are genuine circumstances that may prevent a student from submitting an assessment. If the EC application is successful, there will be no academic penalty for missing the published submission deadline.

Students are normally expected to apply for ECs in advance of the assessment deadline. Students may apply for consideration of ECs retrospectively if they can provide evidence that they could not have done so in advance of the deadline. All applications for ECs must be supported by independent evidence.

Successful EC applications for live oral assessments, including vivas, will result in a deferral of the oral to be organised by faculty, students, and Timetabling for a date as close as possible to the original presentation date. The deadline for supplementary materials, if assigned, will be carried forward by the length of the oral assessment extension.

Missing an oral assessment, including a compulsory viva, without an approved EC will result in a non-submission for the entire assessment and, accordingly, a recorded mark of 0%.

Students are reminded that the ECs procedure covers only short-term issues (within 21 days leading to the submission deadline) and that if they experience longer-term matters that impact on learning then they must contact Student Support and Development for advice.

Under the Extenuating Circumstances Policy, students may defer an assessed element on only one occasion and may request an extension on a maximum of two occasions.

For further information, please refer to the Extenuating Circumstances Policy in the Academic Handbook.

Academic Misconduct

Any submission must be a student's own work and, where facts or ideas have been used from other sources, these sources must be appropriately referenced. The University reserves the right to hold a viva if there are concerns about the authenticity of a student's or learner's work. The Academic Misconduct Policy includes the definitions of all practices that will be deemed to constitute academic misconduct. This includes the use of artificial intelligence (AI) where not expressly permitted within the assessment brief, or in a manner other than specified. Students should check this policy before submitting their work. Students suspected of committing Academic Misconduct will face action under the Policy. Where students are found to have committed an offence they will be subject to sanction, which may include failing an assessment, failing a course or being dismissed from the University depending upon the severity of the offence committed. For further information, please refer to the Academic Misconduct Policy in the Academic Handbook.

Version History

Title: Assessment Brief Template **Approved by: The Quality Team**					
Version number	**Date approved**	**Date published**	**Owner**	**Location**	**Proposed next review date**
4.0	March 2023	March 2023	Registrar	VLE/ Faculty Resources Page	March 2024
3.0	August 2022	August 2022	Registrar	VLE, Faculty Resources Page	July 2023
2.3	December 2021	December 2021	Registrar	VLE	August 2022
2.2	August 2021	August 2021	Registrar	VLE	August 2022
2.1	September 2020	September 2020	Registrar	VLE	August 2021
2.0	September 2020	September 2020	Registrar	VLE	August 2021
1.0	August 2019	August 2019	Registrar	VLE	August 2020
Referenced documents	AQF7 Academic Regulations for Taught Awards; Extenuating Circumstances Policy; Academic Misconduct Policy; Course Syllabus				
External Reference Point(s)	UK Quality Code Theme: Assessment				