**Faculty of Computing, Engineering & Media (CEM)**

**CETC3904 Coursework Brief 2023/24**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Module name:** | **Functional Software Development** | | | | | | | |
| **Module code:** | **CTEC3904** | | | | | | | |
| **Title of the Assessment:** | **Practical Assignment** | | | | | | | |
| **This coursework item is:** ***(delete as appropriate)*** | | | | **Summative** | |  | | |
| **This summative coursework will be marked anonymously: *(delete as appropriate)*** | | | | |  | | | No |
| **The learning outcomes that are assessed by this coursework are:**   1. Analyse and critically review the principal concepts of functional software design 2. Critically evaluate the support for the application of functional software design in the context of a contemporary programming language 3. Apply functional software design to produce a software solution to a practical problem 4. Analyse a given functional software solution in terms of relevant performance criteria | | | | | | | | |
| **This coursework is**: ***(delete as appropriate)*** | | | **Individual** | | | |  | |
| If other or mixed ... explain here: | | | | | | | | |
| **This coursework constitutes** 100% **of the overall module mark.** | | | | | | | | |
| **Date Set:** | | **Monday 26th February 2024** | | | | | | |
| **Date & Time Due (the deadline):** | | **Monday 22nd April 2024 at 12.00 noon** | | | | | | |
| **In accordance with the University** [**Assessment and Feedback Policy**](https://www.dmu.ac.uk/about-dmu/quality-management-and-policy/academic-quality/learning-teaching-assessment/assessment-feedback-policy.aspx), **your marked coursework and feedback will be available to you on:** | | | | | **Tuesday 14th May 2024** | | | |
| You should normally receive feedback on your coursework **no later than 15 University working days after the formal hand-in date,** provided that you have met the submission deadline  If for any reason this is not forthcoming by the due date your module leader will let you know why and when it can be expected. The Associate Professor Student Experience ([CEMstudentexperience@dmu.ac.uk](mailto:CEMstudentexperience@dmu.ac.uk)) should be informed of any issues relating to the return of marked coursework and feedback. | | | | | | | | |
| **When completed you are required to submit your coursework via:**  *There are TWO items to upload to the LearningZone shell. Please refer to the Assignments tab on LearningZone for details. Your submission is not complete without both of these uploads.*   1. *A report must be submitted to**Turnitin via LearningZone****.*** *Place your answers within a copy of the Report Template (CTEC3904 Matrix Report.docx) available on LearningZone. The markers will look for your answers under the matching sections within the report so please do not reorder or rename these sections.* 2. *A copy of your Scala code* ***Matrix.scala*** *must be uploaded to LearningZone.*   ***Only upload your Matrix.scala file****.*  *Do not upload anything else or any other file type. In particular, please: do not upload compiled classes etc. from the IDE; do not upload the test suite (MatrixScalaTest.scala) because we will use our own copy when marking; do not upload the MatrixLab.scala file – this is not used for marking.* | | | | | | | | |
| **Late submission of coursework** **policy:**  Late submissions will be processed in accordance with current [University regulations](https://www.dmu.ac.uk/about-dmu/quality-management-and-policy/academic-quality/academic-regulations-assessment-boards/academic-regs-assessment-board-homepage.aspx).  ***Please check the regulations carefully to determine what late submission period is allowed for your programme.*** | | | | | | | | |
| **Academic Offences and Bad Academic Practices:**  Please ensure you read the section entitled “Academic Offences and Bad Academic Practice” in the module handbook or the relevant sections in this link: [BaseCamp Link: Overview: Assessment and Good Academic Practices](https://eobject.dmu.ac.uk/Basecamp/content/) | | | | | | | | |
| **Tasks to be undertaken: See below** | | | | | | | | |
| **Deliverables to be submitted for assessment: See below** | | | | | | | | |
| **How the work will be marked: See below** | | | | | | | | |
| **Module leader/tutor name:** | **David Smallwood** | | | | | | | |
| **Contact details:** | [**drs@dmu.ac.uk**](mailto:drs@dmu.ac.uk) **(Office: GH6.70)** | | | | | | | |

Should you need any further information or advice please email [cemadvicecentre@dmu.ac.uk](mailto:cemadvicecentre@dmu.ac.uk)

Contents

[1 Download and install 4](#_Toc159858705)

[2 Background 5](#_Toc159858706)

[3a Coursework description (coding) 6](#_Toc159858707)

[3b Coursework description (report) 9](#_Toc159858708)

[4 Marking Scheme 10](#_Toc159858709)

# 1 Download and install

1. Download from **LearningZone** and install into your **IntelliJ** project the Scala class/object source file **Matrix.scala**. It expects to be installed in the package **lib.matrix**:

… -> **lib** -> **matrix** -> **Matrix.scala**

The code inside **Matrix** uses the **Picture** library so you will need to have installed this library in

… -> **lib** -> **picture** -> **Picture.scala**

The details of the Picture library are covered in Unit06. The Picture library is used to create nicely-readable versions of matrices to display on the console.

1. Download from **LearningZone** and install into your **IntelliJ** project the object source file **MatrixLab.scala**. It expects to be installed in the package **lib.matrix**:

… -> **lib** -> **matrix** -> **MatrixLab.scala**

This object contains a collection of matrix definitions and some example operations. The results are printed nicely by converting them to Pictures. It is easier to see the results your operations are generating when the matrices are printed out nicely. Use and adapt this program as your own experimental laboratory. It is provided for you to help you visualize and develop your operations, but it will not be part of the work that you hand in.

1. Download from **LearningZone** and install into your **IntelliJ** project the *ScalaTest* file **MatrixScalaTest.scala**. It expects to be installed in the package **test.scala.matrix**:

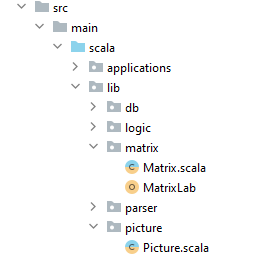
… -> **test** -> **scala** -> **matrix** -> **MatrixScalaTest.scala**

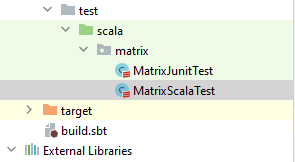
This object contains a suite of Scala test cases (40 in total).

Ensure that *ScalaTest* has been added to your library dependencies within the file **build.sbt**

*libraryDependencies* += "org.scalatest" %% "scalatest" % "3.2.17" % "test"

After the first three steps you should see a file structure similar to that in the picture below. In the picture ignore other libraries that are not relevant here[[1]](#footnote-1)





1. Download from **LearningZone** the file **CTEC3904 Matrix Report.docx**. This is the template for including the reflective analysis and a copy of your code.

# 2 Background

This coursework is concerned with matrices. For a refresher, the lecture slides for a Level 4 module in which matrices are covered has been provided on **LearningZone**. We only require the basic operations on matrices. The concepts you need to look up are:

* matrix identity
* matrix addition
* matrix multiplication
* matrix transposition

The DMU notes above provide sufficient background information. However, there is plenty of other material available online regarding matrices. For example, Wikipedia provides a fairly thorough background: [Matrix (mathematics) - Wikipedia](https://en.wikipedia.org/wiki/Matrix_(mathematics)).  
The sections you need are all in the early part of the article.

# 3a Coursework description (coding)

You have been given a partially-completed Scala library, **Matrix.scala**, that provides a class for matrices, and a companion object containing some useful methods. The file has been organised as follows:

* The class **Matrix** is introduced with a parameter, rows, of type Seq[Seq[Int]]

class Matrix(val rows: Seq[Seq[Int]])

The reason that the parameter to the constructor is of type Seq[Seq[Int]] is because this allows matrices to be constructed from any conformant instance of a subtype: e.g. Vector[Vector[Int]] or List[List[Int]].

* A number of definitions and predicates to do with the dimensions of the matrix are provided. The require statements throw an exception if an incorrectly-dimensioned data structure is encountered.
* A set of methods for defining matrix equality and for printing matrices are provided. The methods canEquals, equals, and hashCode provide the infrastructure for overriding the standard *equals* method, and this follows the scheme described in [How to define an `equals` method in a Scala class (object equality) | alvinalexander.com](https://alvinalexander.com/scala/how-to-define-equals-hashcode-methods-in-scala-object-equality/).  
    
  The toPicture method creates a two-dimensional ASCII image of a matrix using the Picture library.  
    
  Finally, the toString method overrides the top-level toString method, and simply converts the Picture representation into a string for printing. These methods are provided for convenience – you don’t need to change them or add to them.
* A set of matrix methods is provided with the code already completed. You can use these as examples of how such operations might be written. You can also use them in the definitions of your own methods, whenever appropriate. The methods provided are:  
  1. **apply**: looks up an element in a matrix using a row/column index. The first row is indicated by 1 and the first column is indicated by 1 as is conventional for matrices.
  2. **transpose**: transposes a matrix.
  3. **map**: applies a function to every element in the matrix.
  4. **–**(subtract): subtract one matrix from another. This is defined in terms of **+** (add).
  5. **reduceRows**: applies a binary operator to combine corresponding elements in adjacent rows producing a single-row matrix as a result. This method is defined in terms of reduceCols.
  6. **reduce**: applies a binary operator to combine all the elements in the matrix: it is a combination of reduceRows and reduceCols.
  7. **above**: puts two matrices with the same number of columns on top of each other. Thus an *m1 n* matrix above an *m2 n* matrix produces an *(m1+m2) n*  matrix.
* [40%] **You are required to complete the following methods for the coursework**.  
    
  These are:  
  1. **+** (add): adds two matrices together. The two matrices must have the same number of rows and columns.  
       
     If the parameter is an incompatible matrix then the method should throw an IllegalArgumentException.[[2]](#footnote-2)
  2. **\*** (multiply): multiplies two matrices together. The two matrices must be compatible – i.e. an *m n and an n p* are compatible. (E.g. a *3 5* can be multiplied by a *5 4* but a *3 5* is not compatible with a *2 6.*)  
       
     If the parameter is an incompatible matrix then the method should throw an IllegalArgumentException.
  3. **power**: raises a *square* matrix to a given number. If the matrix is not square then an IllegalArgumentExceptionmust be thrown.  
       
     If the number given is negative then an IllegalArgumentExceptionmust be thrown.

Otherwise, *power* behaves thus:

* + - M.power(0) = the identity matrix with the same order as M
    - M.power(n) = M M … M (n times, provided n>0)  
      In particular, this means that M.power(1) = M
  1. **beside**: puts two matrices with the same number of rows beside each other. Thus an *m n1* matrix beside an *m n2* matrix produces an *m (n1+n2)* matrix.  
       
     If the parameter is an incompatible matrix then the method should throw an IllegalArgumentException.
  2. **reduceCols**: applies a binary operator to combine corresponding elements in adjacent columns producing a single-column matrix as a result. For example:

| 1 2 3 4 5 | | 15 |

| 2 4 6 8 10 | | 30 |

| 3 6 9 12 15 | reduce (\_+\_) = | 45 |

| 4 8 12 16 20 | | 60 |

| 5 10 15 20 25 | | 75 |

* 1. **id***: g*enerates the identity matrix with the supplied dimension. Thus,

id(1) = | 1 |

id(2) = | 1 0 |

| 0 1 |

id(3) = | 1 0 0 |

| 0 1 0 |

| 0 0 1 |

etc.  
  
If the supplied parameter is zero or negative then an IllegalArgumentExceptionmust be thrown.

Overall there are six methods to implement.  
  
You are advised to use an incremental development approach (i.e. one method at a time) using the unit tests provided in the class **MatrixScalaTest**.  
  
As this is test-driven development (TDD) you would expect the tests to fail initially, and the number of successful tests to increase as the development proceeds.  
  
There are 40 test cases overall. Look in the file to see each of the test cases.

* [20%] **The quality of your code solutions will be assessed in two ways**:  
  1. *How functional is your code*?  
     We should see:  
     - no use of variable update;  
     - that all functions/methods are referentially transparent;  
     - no use of mutable data structures;  
     - good use of higher order functions.
  2. *How well explained is your code*?  
     We should see good comments before and, wherever appropriate, within the methods you write, to explain your thinking/reasoning. This is particularly necessary when performing a lot of work with a small number of higher order functions/methods which are composed together.

# 3b Coursework description (report)

The report asks you to address a number of aspects. A template document called **CTEC3904 Matrix Report.docx** is provided: please download this and edit it to include your answers.

The aspects you are asked to write about are described below:

1. [20%] **The power method**Copy the code for this method into your report.

Using a combination of written English **and** diagrams, describe in detail your code solution. The English explanation should be clear and concise, highlighting the key features of your solution.  
  
The diagrams should support the text, and should assist in understanding how your algorithm works.  
  
Comment upon any particularly important or noteworthy aspects of your solution.

1. [20%] **The underlying matrix implementation**

val *elements*: Vector[Vector[Int]]

What would be the performance implications of changing this to

val *elements*: List[List[Int]]

**Comment upon the effect on performance** you would expect this to have on each of the following methods – give reasons for your answer in each case:

1. The apply method defined in the Matrix class:  
     
   **def** apply(row: Int, col: Int): Int
2. The beside method defined in the Matrix class:  
     
   **def** beside(other: Matrix): Matrix
3. The map method defined in the Matrix class:  
     
   **def** map(f: Int => Int): Matrix

# 4 Marking Scheme

Each of the components will be marked according to the following marking scheme.

1. **[40%] Code correctness and completeness**  
   The mark will equal the number of Scala Tests that pass. We will use the published set of 40 ScalaTests.  
     
   However, if the code for any methods is not written in a functional style then any passed tests for those methods will not be counted.

|  |  |
| --- | --- |
| 0 | Not submitted or does not compile/run or fails all the tests. |
| : | = the number of tests passed. Remove the score for any passed tests related to methods that use non-FP techniques. |
| 40 | Passes all tests, and all methods are implemented using only FP techniques |

1. **[20%] Code quality and commenting**  
   **[10%] Code quality (evidenced by the code).**We are looking for appropriate use of functional features including appropriate use of higher order functions.  
     
   We will take a holistic look at all the code you have written and make a balanced judgement on the overall impression. We want to see neither mutable state nor updates to mutable state.[[3]](#footnote-3) We will use a grade between 0 and 5:

|  |  |
| --- | --- |
| 0 | Not submitted or does not compile/run. |
| 1 | No, or only isolated, functional programming features used showing a very limited understanding, below the threshold to pass. |
| 2 | Some FP features have been used that demonstrate a basic understanding in some areas. |
| 3 | Only FP features have been used, and to a level that demonstrates a clear understanding of key concepts. |
| 4 | Excellent use of FP features that demonstrates an authoritative grasp of the material. |
| 5 | Very sophisticated use of FP features that demonstrates an exceptional level of understanding. |

**[10%] Commenting (in the code)**.  
  
We are looking for concise, but helpful, comments. These are very likely to appear before a method to describe the overall approach. They may also occur within the method code to describe code that is quite complex or sophisticated (e.g. combinations of higher order functions).  
  
We are looking for code that is highly *readable* by another programmer. This criterion is necessarily subjective and will, therefore, be based upon academic judgement. Therefore, aim for clarity and conciseness so that the marker can see easily your intention.

We will use a grade between 0 and 5 and a mark will be allocated from the following table that the marker judges fits the category more closely than the others.

|  |  |
| --- | --- |
| 0 | Not submitted or effectively absent. |
| 1 | Only a small number of comments that may not be particularly helpful and/or obvious gaps where comments are needed. |
| 2 | Some comments are used but they are mostly perfunctory or contain insufficient detail to describe the code they are attached to. The comments may contain misunderstanding of how the FP concepts work, or they may be convoluted, or even contradictory in places. |
| 3 | Good use of comments. They are mostly provided where they aid understanding of tricky or dense code. There may be some less relevant comments and the overall style may be somewhat inconsistent. |
| 4 | Excellent use of comments. For the majority of the cases the comments are clear and concise and document the features used appropriately. The level of explanation shows an excellent understanding of the features. |
| 5 | Exceptional use of comments. The style is informative (even inspired), consistent, easy-to-read, and concise, without missing any important detail. The comments make extremely insightful remarks that demonstrate outstanding insight and/or flair. |

1. **[20%] Explanation of the power method. (In the report).**

We will use a grade between 0 and 5 and a mark will be allocated from the following table that the marker judges fits the category more closely than the others.

|  |  |
| --- | --- |
| 0 | Not submitted or effectively absent. |
| 1 | Code missing from report and/or poor description and/or lack of effective supporting diagrams. The solution does not meet the minimum threshold of understanding required. |
| 2 | Code has been included. The English description reflects reasonably well the action of the code and, together with the accompanying diagrams, the answer shows a basic understanding that meets the minimum threshold. The solution is likely to be not particularly efficient. |
| 3 | Code has been included. The English description reflects well the action of the code and, together with the accompanying diagrams, the answer demonstrates a sound understanding. The code may be a basic solution which has not been made especially efficient. |
| 4 | Code has been included. The English description is thorough and concise and fully describes the action of the method. The accompanying diagrams support the explanation very well. An excellent level of understanding is evidenced by the description, and the method itself is written with efficiency in mind. |
| 5 | Code has been included. The English description is thorough and concise and demonstrates outstanding insight and/or flair. The accompanying diagrams are excellent. A very high level of understanding is evidenced both by the description and by the implementation itself, in which an efficient solution has been explained and executed. |

1. **[20%] Discussion of the implications of changing Vector to List in each of the three given methods: *apply*, *beside*, and *map*. (In the report).**  
     
   We will use a grade between 0 and 5 and a mark will be allocated from the following table that the marker judges fits the category more closely than the others.

|  |  |
| --- | --- |
| 0 | Not submitted or effectively absent. Or the discussion demonstrates a significant misunderstanding of the implications of the substitution. |
| 1 | The solution might not mention all three methods or does so but the effect of the substitution of Vector for List has been explained properly in **only one** **method**. |
| 2 | The solution covers all three methods. The effect of the substitution of Vector for List has been explained properly in **two** **of the methods**. The explanation is basic and may have flaws, but demonstrates sufficient understanding to meet the basic threshold. |
| 3 | The solution covers all three methods. The effect of the substitution of Vector for List has been explained properly in **two of the methods**. The explanation is good and contains only minor errors. |
| 4 | The solution covers all three methods. The effect of the substitution of Vector for List has been explained properly in **all three methods**. The explanation is excellent and demonstrates a full appreciation of the factors involved. |
| 5 | The solution covers all three methods. The effect of the substitution of Vector for List has been explained properly in **all three methods**. The explanation is outstanding and demonstrates significant flair and insight. |

1. We added a Junit test file only as a template. Look on the **LearningZone** shell for details of how to add a Junit test suite if you would like to use one for testing. For marking, we will use the *MatrixScalaTest* suite. [↑](#footnote-ref-1)
2. You can use the *require* method to assert the required property – see the earlier methods for examples of how to use *require*. [↑](#footnote-ref-2)
3. Do not import (or write to) **mutable** data structures (this includes any instances of the default Array class); and do not use **var** to declare any variables. [↑](#footnote-ref-3)