Faculty of Computing, Engineering & Media (CEM)
CETC3904 Re-sit Coursework Brief 2023/24

Module name:	Functional Software Development		
Module code:	CTEC3904		
Title of the Assignment:	Practical Assignment		
This coursework item is: (delete as appropriate)		Summative	
This summative coursework will be marked anonymously		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	
This coursework constitutes 100% **to the overall module mark.**			
Date Set:	Tuesday 2nd July 2024		
Date & Time Due:	Friday 16th August 2024 @ 12:00 (noon)		
Your marked coursework and feedback will be available to you on: 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) should be informed of any issues relating to the return of marked coursework and feedback. Note that you should normally receive feedback on your coursework by **no later than four working weeks after the formal hand-in date,** provided that you met the submission deadline.			On or before 10th Sept. 2024 *(These are indicative marks subject to moderation and ratification at the September Assessment Board)*
When completed you are required to submit your coursework to the Turnitin portal.			
Late submission of coursework policy: Late submissions will be processed in accordance with current University regulations. **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
Tasks to be undertaken: See (following) attached document.
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 (GH6.70)**

Important note on cheating

Please **do not** be tempted to search the internet for an existing solution and then hand this in – e.g. if you run out of time. This is **cheating** – it is better to hand in what you have honestly achieved by yourself than try to get credit for someone else's work and risk getting caught. Cheating is an **academic offence**.

Do not use anyone else's material without referencing it – this is **bad academic practice** or **plagiarism**. These are considered as **academic offences**.

Do not work jointly on a solution; do **NOT** give your solution to anyone else to "help them" and **do not** accept anyone else's solution as "guidance". Such practice could lead to an allegation of **collusion** which is an **academic offence**. **All parties** involved in collusion (the givers, the receivers, the collaborators) can be found guilty of an academic offence, irrespective of motive.

Overview

This piece of work requires you to produce a self-contained program written using functional programming techniques in Scala and to submit your program along with a commentary – see the section (below) **What the report should contain**.

The work must be presented within a **single Word document** – and this includes a copy of the Scala code. Please read and follow carefully the details below of how to include the code in the document.

Scenario

Invent your own scenario and produce piece of **Scala code** that demonstrates a solution to the problem you have invented. The purpose of your example is to demonstrate the correct use of some distinguishing features of functional programming (FP).

The example code that you produce should include at least one clear example of each of the following FP aspects in Scala. Furthermore, the use of each of the following aspects should be used appropriately within the solution.

1. The use of **higher order functions**
2. The use of **partial application**
3. The use of **case classes** and/or **case objects**
4. The use of **pattern matching**
5. The use of **recursion that terminates** – i.e. not an infinite loop

You have freedom in this assignment to invent the problem and the solution. Remember, the primary purpose of the code example is to demonstrate the FP features outlined above. However, the example will work better if the scenario is credible.

Please think carefully about the scenario you are going to construct. It must be your own. Do not share ideas with anyone else otherwise your solutions may appear similar when checked by Turnitin. Do not (re)present code obtained from other sources or from the lecture notes. **This program must be your own, individual idea constructed by you** to demonstrate the FP features outlined above.

You may use any of the standard Scala classes in your solution. So, for example, you could use instances of Scala Lists. However, **only use standard Scala libraries and instances from them.**

Restrictions

Because you are constructing an illustrative example, the code must observe the following restrictions. You will need to take care to ensure that the presentation conforms to the stipulations below.

1. The code should be as many lines as necessary to achieve the goal set. However, as a guide, the code should probably be around 60 lines (including blank lines and braces). There will be no penalty applied for submitting longer programs but there are no extra marks for unnecessary verbosity. Try to be concise.
2. No line of code should line wrap when presented in your Word report. It is possible

to write very long lines in an IDE which are then wrapped when pasted into Word. You may need to reformat your code or adjust the margins of your word document to meet this restriction. If you submit code with long lines that wrap around then it is harder for the marker to read and appreciate your solution. It is in your interest to get your message across as clearly as possible so please format your code neatly as described.

3. The code should be laid out using whitespace and indentation appropriately – remember this is an example of good practice to demonstrate FP to others.
4. The program must be self-contained. It must include the object wrapper, any functions, classes, methods, etc., and a main program. The program should not rely on any external files or resources.
5. You should include any comments within the code that are necessary to interpret specific aspects that may not be obvious to another person (the assessor). However, restrict such comments to those that are crucial, because the majority of the explanation will be provided by you later in your report.
6. It must be possible for the assessor to copy/paste it directly into an IDE and for it to compile and run without any editing or reformatting required. You might want to ensure that this procedure works before you submit the work.

What the report should contain

Your report should contain exactly the following sections. We have provided the structure and the headings that you should use in your submission. Do not change the structure of the document. We recommend that you use font Calibri size 12 for the text (and font size 10 for the code).

Your report should have the following structure including section titles.

(If you have nothing to say in any of the sections then keep the heading and write the words "no answer" instead of an answer in that section).

Section 1: Scenario

In this section you should describe briefly what the program is going to do. What task it is solving?

This section will probably be one or two paragraphs.

Section 2: Code

In this section you should include your code copied/pasted from the IDE you are using into your Word document.

You may have to change the font size (recommend 10 point for code) so that it fits nicely on

the page without any line-wrapping occurring. Remember you can widen the margins in Word if necessary.

Below is an example piece of code taken from the lab exercises. Note how the whole program is here: the surrounding object; the local function definitions; and the main function. (If you use a proportional font like Calibri as shown below then you can get more code on a single line without wrapping than if you use a fixed width font).

In place of the example below you would copy/paste your own program here:

```
object Fun03 {

 def averageWordLength(words: List[String]): Int = {
  var count = 0
  var chars = 0
  words foreach { word =>
   count += 1
   chars += word.length
  }
  return chars/count
 }

 def main(args: Array[String]): Unit = {

  val sentence = List("I", "like", "writing", "functional", "programs", "in",  "Scala")

  println( averageWordLength( sentence ) )

   }
}
```

Section 3: Analysis

In this section you should explain how each of the aspects that were required to be demonstrated have been included in your program.

Use the following subsections to do this. *Each subsection will probably be around two or three paragraphs – feel free to add code snippets or diagrams if these help you to explain the point you are making.*

3.1 The use of higher order functions

In this subsection you should point out where in your code you have used **higher order functions**. You should also explain what role they are playing in your solution.

*In this section it is your task to convince the assessor that you understand **where the higher order functions are used in your code**; and that **they have been used appropriately**.*

3.2 The use of partial application

You have been asked to include at least one example of **partial application**. You should

point out where in your code you have used partial application. You should then choose one instance of this (perhaps the most prominent) and explain clearly why this is a partial application and what the partial application achieves in this context.

*Your task is to convince the assessor that you can **identify partial application correctly** in your code; and that you understand **why it is a partial application**.*

3.3 The use of case classes

You should identify all the **case classes and/or case objects** that you have used in this subsection and explain the purpose each plays in your solution.

*Your task is to convince the assessor that you can **identify correctly the case classes/case objects**; and that you have **used them appropriately**.*

3.4 The use of pattern matching

You should indicate where in your program you have used **pattern matching**. You should describe the different kinds of pattern matching you have used and why you have used those specific patterns in each case.

*Your task is to convince the assessor that you have **used pattern matching correctly**. Furthermore, your examples should demonstrate the **use of pattern matching over case class/case object instances** - not just over simple data types.*

3.5 The use of recursion

You should indicate where you have used **recursion**. You should explain clearly how the recursion makes progress towards a terminating condition and you should also explain clearly where the termination condition is tested.

*Your task is to convince the assessor that you understand how to **use recursion to perform iteration**; that you can **explain the termination condition**; and that you can **demonstrate how each call makes progress towards the terminating condition**.*

Assessment criteria

The work will be assessed using the decision tree below.

Has the work been submitted on time?

1. **No**. Do not mark the work and return a mark of zero. *Re-sit work has to be submitted on time.*

2. **Yes**. Does the program **compile** and **run**? (This will be tested by copy/paste from the report into an IDE. The code should compile and run following this procedure.)
    a. **No**. *Then the assignment is failed. Without a working program the learning outcomes for the module have not been met. To establish a mark for the work, treat the analysis sections as independent discussions on each criterion and mark with the following scale.*

Use a scale (0 - 7) to assess each of the five analysis sections in respect of the program listing as presented in the report. (0=no answer; 1=irrelevant or mostly incorrect; 3=partially correct; 5=mostly correct; 7=correct). Choose the point on the scale that fits the answer more closely than the others and interpolate as necessary. Sum these to a maximum of **35**.

In the assessor's opinion and only by looking at the code presented, does the program appear to be closer to a working solution or further away from one, despite not working?

i. **Further away**. No further marks to be obtained.

ii. **Closer to working**. Add **three** marks.

b. **Yes**. *A working program has been submitted.*

Read Section 1 and run the program. Does the program when run provide a reasonable solution to the problem as described?

I. **No**. This partially fulfills LO3. If the code performs some other related but reasonable function then award **25 marks** for the working code. If the code only performs some simplistic and/or disjointed function then award **15 marks**. If the program is trivial or perfunctory then award **5 marks**.

Then proceed with part iii (below) awarding marks for the five individual analysis sections.

II. **Yes.** This fulfils LO3: award **35 marks**. LOs 1, 2 & 4 are covered jointly by the analysis sections below.

Then proceed with part iii (below) awarding marks for the five individual analysis sections.

III. Use a scale (0 - 13) to assess each of the five analysis sections in respect of the program listing as presented in the report. (0=no answer; 2=irrelevant or mostly incorrect; 5=partially correct; 8=mostly correct; 11=correct; 13=correct and shows particular flair or insight). Choose the point on the scale that fits the answer more closely than the others and interpolate as necessary. Sum these (maximum = **65)**.

Tip

If you can't get a fully working program with all of the aspects covered then it is better to produce a smaller program that covers fewer aspects than to submit a program that doesn't work. Non-working programs cannot pass this re-assessment, especially because you are the architect of both the problem and its solution. This is completely in your own hands.