Faculty of Technology – Course work Specification 2018/19

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:	Monday 15th January 2019
Date & Time Due:	Monday 1st April 2019 no later than 2pm
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 Head of Studies (headofstudies-tec@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.	8th May 2019

When completed you are required to submit your coursework to:
- Blackboard VLE through an assignment submission portal

Late submission of coursework policy: Late submissions will be processed in accordance with current University regulations which state:
"the time period during which a student may submit a piece of work late without authorisation and have the work capped at 40% [50% at PG level] if passed is ***14 calendar days****. Work submitted unauthorised more than 14 calendar days after the original submission date will receive a mark of 0%. These regulations apply to a student's first attempt at coursework. Work submitted late without authorisation which constitutes reassessment of a previously failed piece of coursework will always receive a mark of 0%."*

Academic Offences and Bad Academic Practices: These include plagiarism, cheating, collusion, copying work and reuse of your own work, poor referencing or the passing off of somebody else's ideas as your own. If you are in any doubt about what constitutes an academic offence or bad academic practice you must check with your tutor. Further information and details of how DSU can support you, if needed, is available at: http://www.dmu.ac.uk/dmu-students/the-student-gateway/academic-support-office/academic-offences.aspx and http://www.dmu.ac.uk/dmu-students/the-student-gateway/academic-support-office/bad-academic-practice.aspx	
Tasks to be undertaken: See (following) attached document.	
Deliverables to be submitted for assessment: See (following) attached document.	
How the work will be marked: See (following) attached document.	
Module leader/tutor name:	**David Smallwood**
Contact details:	**drs@dmu.ac.uk (GH6.71)**

Important note

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**.

If you get stuck then please ask the module tutors for assistance and come to the labs – we will be providing help in the labs (but not actually doing it for you of course).

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.

Requirement

A program is required to analyse a Scala source file and produce a concordance of the Scala keywords used in the file. The output should consist of an alphabetical list of the Scala keywords identified in the file, one per line, and next to each keyword should be the number of occurrences of the keyword in the file, followed by a comma separated list of line numbers on which the keyword appears. The list of line numbers should be in ascending numerical order and contain no duplicates. Thus, if a keyword appears more than once on a line then each instance contributes to the overall count, but the line number is only referenced once. For example:

```
Keyword    count     lines
val        12        5, 6, 10, 11, 20, 21, 22, 23, 24, 25, 40,
                     42
var         5        7, 8, 9
while       3        15, 36, 55
```

You should call your program `Concordance.scala`

There is a certain amount of freedom for you to be creative on how the output is presented within the constraints already stated. However, there are some ideas you could think about including: line number ranges for consecutive ranges, e.g. 20-25; an option to specify the maximum width of the output page - this will affect how you word-wrap long lists of line numbers.

As for reading the file you should allow the user to supply a command line argument containing the file name to be used.

Although you are free to be creative with the way the output of your program looks you should not concentrate on adding fancy features to this at the expense of writing a neat, functional solution. This exercise is an opportunity for you to practice and demonstrate what you have learned about writing programs in a functional style. We will expect to see lots of list processing and use of higher order functions. As a rough measure for how well you have used functional techniques in your solution try the following experiment: run your program with its own source code as input - the count for the keyword `var` should be very low - the lower the better. You have done extremely well if the `var` count is zero!

Finally, you should provide a brief discussion about what you have learned about functional programming (FP) by undertaking this exercise. Include this commentary as a Scala comment at the top of your program source file. Avoid writing vague platitudes, but evidence your claims by referring to specific examples of functional techniques you have used and appreciated in your own solution. We are looking to gauge the depth of your understanding from your analysis. We would expect this to be about a few paragraphs long.

Notes on coding style

- We are looking for more FP and less OO. For example, transforming lists using maps and filters, using comprehensions, etc, are FP coding concepts. If you use the data structures in `scala.collection.immutable` then these will be pure FP. Whereas defining an instance of a (mutable) hashmap, (say) `m = new java.util.HashMap()` , and using it as one must (with side effects), `m.put(k,v)` , is an OO approach not an FP one because it causes the mutation of a data structure.
- Using mutable variables in traditional control structures like while loops is very non-FP: `var c = 0; var x = 0; while (c<10){x=x+c; c=c+1}` Whereas writing `val x = (1 to 10) sum` is very FP
- Put useful comments in your code. FP can be very concise – even dense – and needs some explanation. Avoid stating the obvious ("adds one to each value in the list") as this is usually just clutter. However, if you have some interesting higher-order function applications composed then these will probably require some discussion in a related comment. Your program needs to be readable by a human as well as a computer.

Assessment grid

		None	Poor	Weak	Pass	Good	V.G.	Excel.	Outst.
		0	14	29	43	57	71	85	100
Completeness	60%								
Coding	30%								
Analysis	10%								

Completeness

We are looking for a working solution. To what extent does it solve the problem set?

A non-compiling program	None
A program that does something useful but is a long way from a solution	Poor
A program that does a few useful things but does not produce a concordance	Weak
A program that produces an incomplete concordance	Pass
A program that produces a concordance with correct counts for each keyword	Good
A program that produces a concordance with counts and line numbers	V.G.
Meets the specification and demonstrates flair with output presentation	Excel
Exceeds the specification in some outstanding way	Outst

Coding

To what extent does the solution demonstrate an understanding of FP?

A non-compiling program	None
Too little code to really make a judgement	Poor
Essentially a traditional/OO program solution	Weak
A hybrid OO/FP solution - mostly OO/traditional	Pass
A hybrid OO/FP solution - roughly equal measure	Good
A hybrid OO/FP solution - with notable emphasis on FP style	V.G.
A substantially FP solution that fully demonstrates understanding of FP	Excel
A fully FP solution (no use of `var` anywhere other than as data)	Outst

Analysis

To what extent does the analysis describe the FP components, and how well?

No analysis	None
A few vaguely relevant phrases	Poor
Maybe a single poorly-argued paragraph	Weak
Reasonable points made but lacking in substance	Pass
Reasonable points with understanding but lacking backup from code	Good
Good points made backed up with examples from program code	V.G.
Excellent analysis showing real insight - backed up with examples	Excel
Outstanding insight into FP techniques argued from perspective of code	Outst