`

# FUTURE OF SPREADSHEET PROGRAMMING

## A COMPARISON BETWEEN TWO PROMISING TECHNOLOGIES

by

Georgi Ivanov Zhivkov

AALBORG UNIVERSITY
DENMARK

Master thesis

Supervisors: Bent Thomsen and Thomas Bøgholm

.

# SUMMARY

This project deals with a comparison of two different approaches to define user-defined functions (UDFs) in a spreadsheet. One of the ways is Excel's Lambdas and the other is SDFs (sheet-defined functions) in FunCalc.

The project answers three questions. The first question is about the speed of the calculation of each technology and aims to determine which is better for computations. To answer this a number of benchmarks is done. The number of samples for the testing is found by firstly doing a random sampling with the size of 100 and then applying Cochran's formula. The results of these benchmarks were found to be inconclusive but pointed towards the dominance of FunCalc. Excel performed better on two of the 7 benchmarks. One of the sheets that Excel performed better on was "Groundwater_daily" which is a big workbook with multiple sheets. The results for FunCalc on this sheet were consistent with another study. For future work investigation into whether Excel uses parallel or sequential execution also whether the difference between the representation of data in both technologies impacts their performance.

The second question is "which of these technologies is preferred by the programmers" and deals with issues connected to the Cognitive Dimensions. The question is answered by doing an analysis of the Cognitive Dimensions and then a user study involving live coding with people unfamiliar with the technology, which also connects the Discount Evaluation method with the Cognitive Dimensions questionnaire. The result of both the analysis and the user study shows that Excel's Lambdas are more useful compared to SDFs in FunCalc. Participants pointed much more issues in FunCalc than in Excel. Some of the issues of FunCalc that were pointed out were connected to the way of defining the functions, using the cell identifiers instead of variable names, the length of the function definitions, and writing the functions in the function sheet. In Excel, the participants had problems with the parentheses and commas, the name manager window, and the way how the expression comes directly after the parameters.

The third question is whether these technologies have any application in the educational sphere. Are the UDFs applicable in teaching Computational Thinking to people unfamiliar with it? Due to the delimitations of the study only a philosophical discussion on the topic was done. The study looked into the application of functional programming and its similarities to UDFs.

In conclusion, the study proved successful and contributed to the following things:

1. Tested the performance of the recently introduced UDFs in Excel against FunCalc's SDFs.

2. Compared Excel's UDFs to FunCalc SDFs in a study with participants unfamiliar with the technologies and concluded which one is familiar.
3. Investigated how functional programming with spreadsheets can be used to facilitate Computational Thinking.

# ACKNOWLEDGMENTS

I want to thank my family and my friends for believing in me. I want to thank my supervisors for giving me helpful feedback and good suggestions. Finally, I want to give my thanks to the participants of my study for giving me meaningful results for the study.

# ABSTRACT

The goal of this study is to compare two different technologies for creating user-defined functions in Spreadsheets, one of them is the recently introduced Lambda expressions in Excel and the other is the SDFs in FunCalc, and how they can be used to facilitate computational thinking. In order to do so first, a performance test will be conducted and then a user study will be done.

# TABLE OF CONTENTS

# TABLE OF FIGURES

# CHAPTER 1. INTRODUCTION

## 1.1. BACKGROUND

Spreadsheets are computer applications used for data analysis and data storage created more than 40 years ago. The spreadsheets are used in accounting, bioinformatics, and other fields. They are a 2D grid of cells where every cell has coordinates and can contain a value or formula. Their use can be both as a programmable tool for doing calculations and a database.

Spreadsheets can be viewed as a program, where instead of coding in lines, the relationships between data are connected in the aforementioned grid environment (Georgi Zhivkov, 2021). Some of the spreadsheets come with pre-defined functions for arithmetic or statistics and with the option for user-defined functions, for example, Microsoft's Excel and VBA functions.

While there are other data analysis tools, spreadsheets are still one of the most popular ones with 800 million of active users Microsoft Excel (Gislason, 2018). But, according to some data scientists, spreadsheets are falling behind in the age of Big Data (Chase, 2020). And thus finding an improvement of the support for the end-user development can keep the spreadsheets' status as a useful and programmable tool used by many data scientists around the world since their current implementation provides little support for abstraction and reuse of computations, unless external languages like VBA are used (Sestoft, 2017).

## 1.2. PROBLEM STATEMENT

In the age of big data, the need for better, robust, and reliable calculations inside spreadsheets is increasing in order to keep their relevance. One of the ways that it can be achieved is the user-defined function approach since these functions offer better abstraction and can offer better programmability. In this project, we will explore two promising solutions that allow user-defined functions in spreadsheets and compare them on several criteria. One of them being the recently introduced Lambdas in Excel and the other being the FunCalc technology, creation of Peter Sestoft.
The study aims to answer thtree main questions:
Q: Which of the methods has better performance?

The answer to this question is found in the performance testing that is performed. The performance testing looks at criteria such as speed, memory usage, and others.

Q: Which of the methods is more familiar and understandable to the users?

This answer is found by conducting a user study and analyzing the results and doing own analysis of chosen cognitive dimensions.

Q: Can UDFs languages be used to facilitate computational thinking?

To answer this question, a philosophical study based that considers the answers from the user study will be done.

## 1.3. INTRODUCTION OF THE TECHNOLOGIES

Currently, Excel uses an external language called Visual BASIC for Applications for creating user-defined functions. In order to avoid the drawbacks of VBA, Microsoft is introducing Lambda expressions. One of the first proposals for creating user-defined functions was in (Simon Peyton Jones, 2003). There the authors argue that adding one of the most important mechanisms- the ability to create reusable abstractions/user-defined functions will be beneficial to the Excel users. According to them, the functions help the user to: Reduce errors during maintenance, for real estate management, achieve better performance, protect intellectual property and encapsulate and re-use domain-specific expertise.

The first technology that is discussed in the report is the FunCalc. FunCalc is the extension of CoreCalc which is an environment that has the core functionality of spreadsheets. Funcalc allows user-defined functions in the form of “sheet-defined functions” without resorting to external languages such as VBA. Both of the aforementioned technologies are research prototypes and not fully usable replacements for Excel and other spreadsheets (Sestoft, 2017).

The following screenshot shows the FunCalc environment. The FunCalc environment can create both function sheets and non-function sheets.

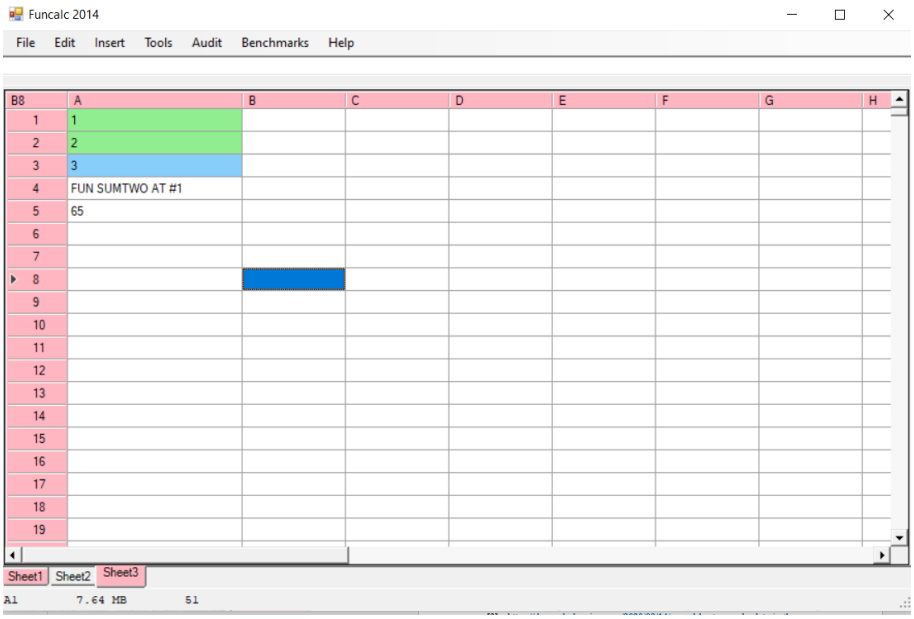


*Figure 1. Overview of the FunCalc GUI*

Unlike Excel, the functions can only be defined inside the spreadsheet and it doesn't support "Name Manager" functionality. The functions are defined with the "DEFINE" keyword. The sheet-defined functions are compiled to .NET(CLI). Funcalc supports some of the basic formulas from Excel's formula language such as "SUM", "AVERAGE" etc. It also supports different types of Arrays such as Horizontal (HARRAY) and a Vertical (VARRAY) while Excel doesn't make difference between these.

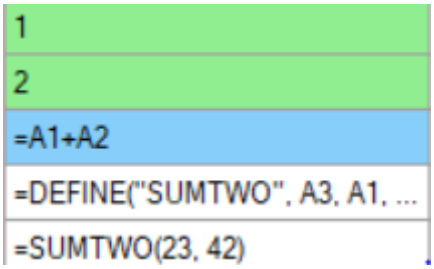


*Figure 2. Example formula*

The second technology that is introduced in this project is Excel's Lambda functions. Excel's Lambda functions made the language Turing-complete since Lambda calculus

is Turing-complete. and gave it the optionality for user-defined functions. Currently, Lambda functions are available only for beta testers. Like many functional-paradigm languages, Excel's Lambdas come with helper functions.

The first step of creating an Excel Lambda function is opening the Name Manager, where user-defined Lambdas can be named.

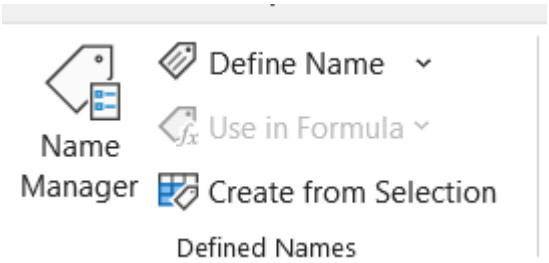


*Figure 3. Picture showing where the name manager menu is located.*

By opening the Name Manager the user is prompted to the following window, where they can choose to either create, edit or delete a named function.

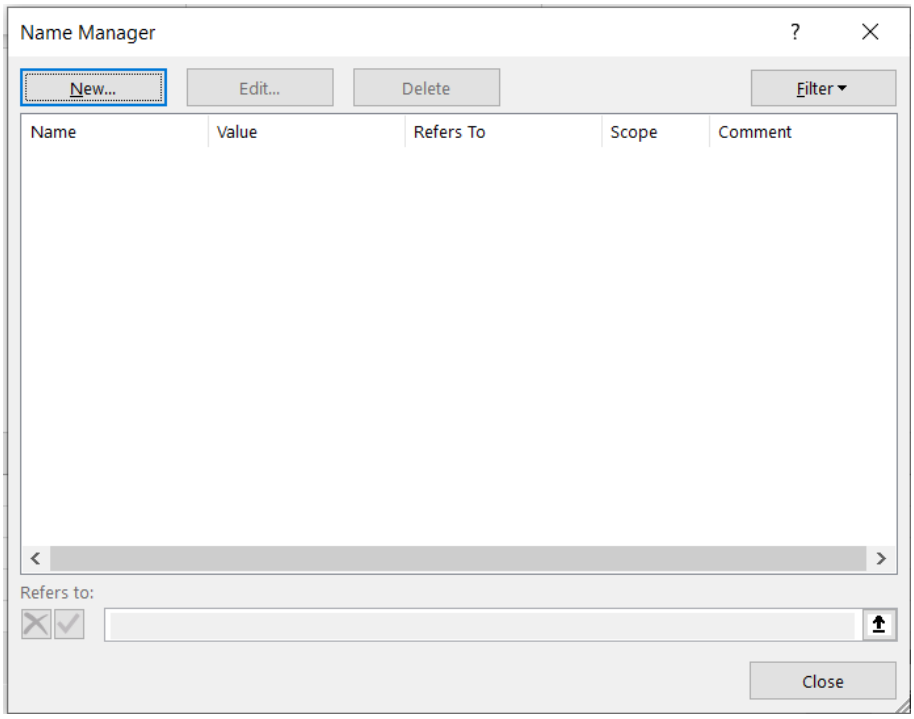


*Figure 4. Name manager window*

After deciding to create a new function the following window appears, it allows the user to name a function and define it, by writing it in the “refers to” box. Another way that  Lambdas can be named is by the use of the LET keyword

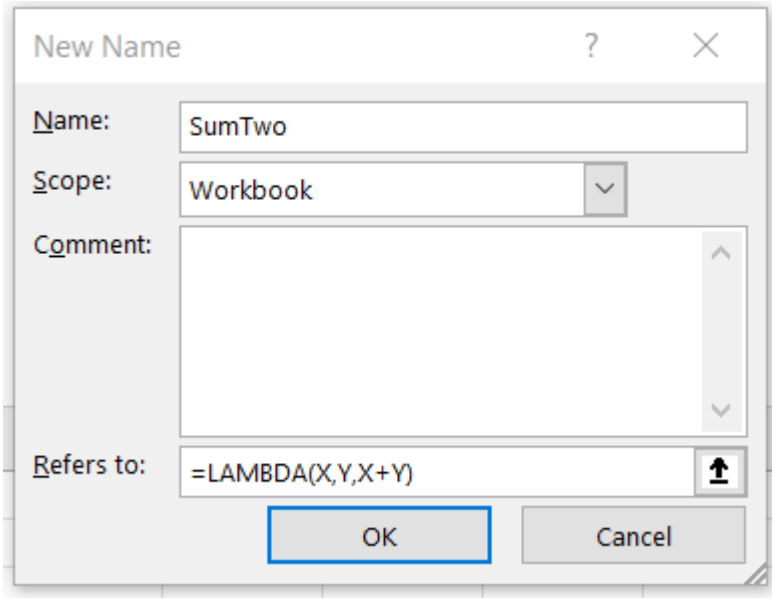


*Figure 5. Creating a new named function*

The Excel's Lambda's come with implementation for standard helper functions.

| | |
|---|---|
| **MAP** | Returns an array formed by "mapping" each value in the array(s) to a new value by applying a lambda to create a new value. |
| **REDUCE** | Reduces an array to an accumulated value by applying a LAMBDA function to each value and returning the total value in the accumulator. |
| **SCAN** | Scans an array by applying a LAMBDA to each value and returns an array that has each intermediate value. |
| **MAKEARRAY** | Returns a calculated array of a specified row and column size, by applying a LAMBDA function. |

| | |
|---|---|
| **BYROW** | Applies a LAMBDA to each row and returns an array of the results. |
| **BYCOL** | Applies a LAMBDA to each column and returns an array of the results. |
| **ISOMITTED** | Checks whether the value is missing, and returns TRUE or FALSE. |

*Table 1. List of Excel's LAMBDA helper functions (Gross, 2021)*

## 1.4. CONSTRAINTS OF THE STUDY

This study has some constraints attached to it. The first of them is the time limitation, the study is conducted in the period 01.02.2022-10.06.2022. The second constraint is finding people fitting the criteria to perform the user testing on.

## 1.5. TARGETED GROUP FOR USER TESTING

The candidates for user testing are people who are familiar with spreadsheet technologies because the study will demand the users to already have some knowledge of the Formula language. It is preferable that some of the users are not from computer science backgrounds since a more extensive study on how the SDFs can be used to facilitate computational thinking can be done.

# CHAPTER 2. PERFORMANCE EXPERIMENTS

## 2.1. PERFORMANCE EXPERIMENTS DECISIONS

The performance testing is performed on some of the programs described in (Bock, 2019), the decision behind it is that these programs both can be used to describe the expressivity but also can be used as a good performance benchmark. Some of these programs are also short and easily implementable thus meaning that they could also be used for user testing. The performance experiments will be done using the external VBA timer function for Excel and by the built-in benchmark for FunCalc.

## 2.2. CHOICE OF PROGRAMS

Since Excel does not support single-celled arrays and array-slicing some of the programs mentioned in (Bock, 2019) are reworked and not translated 1:1. In general, for the study, it's better to create some small and widely used programs.

The first functions that are tested are Fibonacci and Factorial. This decision was motivated by the fact that both programs can be used in the user study later and they are similar in implementation. The Factorial program is used later in the user study since it is a program that uses recursion and recursion and/or looping are concepts that are some of the key components in most programming languages.

Another function is the SUMCOLUMNS. This program contains recursion over an array. The program sums all the numbers in a horizontal array.

Another function that is used the Finding the index of the minimum element. This program is a simple traversing of an array and comparison. The program is easy and simple to implement. A supplementary example for it could be INDEXOF, which finds the index of the first occurrence of an element in the array.

Another book of spreadsheets that have been tested is "Groundwater_daily", it is a prepared data consisting of three sheets, one with 7700 rows of data, one with 8680 rows, and one with 15234 rows.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Data source | | | | | | | | | | | | | | |
| 2 | | | | | | | | | | | Surface elevation (feet): | | 99.44 | | |
| 3 | | | | | | | | | | | | | | | |
| 4 | | | | Latitude | | | Longitude | | | | | | | Delta | | |
| 5 | State | County | deg | min | sec | deg | min | sec | Designation | Date | Level | Elevation | Average | Low | High |
| 6 | FL | Orange County | 28 | 22 | 10 | 81 | 35 | 26 | NAD27 | 1969-01-18 | 93.90 | 5.54 | -2.83 | -6.04 | 1.44 |
| 7 | FL | Orange County | 28 | 22 | 10 | 81 | 35 | 26 | NAD27 | 1969-02-24 | 95.56 | 3.88 | -1.17 | -4.38 | 3.10 |
| 8 | FL | Orange County | 28 | 22 | 10 | 81 | 35 | 26 | NAD27 | 1969-02-25 | 95.55 | 3.89 | -1.18 | -4.39 | 3.09 |
| 9 | FL | Orange County | 28 | 22 | 10 | 81 | 35 | 26 | NAD27 | 1969-02-26 | 95.53 | 3.91 | -1.20 | -4.41 | 3.07 |
| 10 | FL | Orange County | 28 | 22 | 10 | 81 | 35 | 26 | NAD27 | 1969-02-27 | 95.52 | 3.92 | -1.21 | -4.42 | 3.06 |
| 11 | FL | Orange County | 28 | 22 | 10 | 81 | 35 | 26 | NAD27 | 1969-02-28 | 95.50 | 3.94 | -1.23 | -4.44 | 3.04 |
| 12 | FL | Orange County | 28 | 22 | 10 | 81 | 35 | 26 | NAD27 | 1969-03-01 | 95.48 | 3.96 | -1.25 | -4.46 | 3.02 |

*Figure 6. Groundwater Daily*

A sheet defined functions that “mask” the Excel formulas in the fields “Average”. “Low”, and “High” is done. What “masking” means in this context is that the formula from these fields is taken and is just being named as shown below

=AVERAGE(L$6:L$15234)-L15221

This formula is masked i.e. turned into a function as the following Excel function named AVERAGE.

```
=LAMBDA(X,Y,AVERAGE(X)-Y)
```

*Figure 7. Masking with Excel Lambdas*

In a similar fashion the function has been defined in FunCalc:

| | |
|---|---|
| =AVERAGE(A1)-B1 | |
| =DEFINE("avg", A2, A1, B1) | |

*Figure 8. Masking a function in FunCalc*

Performance test on each of the maskings.

## 2.3. HOW THE TESTING IS DONE

The testing methodology is using doing a random sample with the size of 100 and then using the Cochran’s formula to find a suitable sample size for each of the programs as done in (Anne Benedicte Abildgaard Ejsing, 2021). For the Excel a VBA script was written, a modified version of the code taken from (Anon., n.d.) that incorporated the Cochran’s formula (code can be seen in Appendix C), and for FunCalc a modified version benchmark functionality that incorporates the Cochran’s formula was used (code can be seen in Appendix D). The testing was done on the

latest version of Excel up to 01.04.2022 and the publicly available version of FunCalc from 2014.

## 2.4. TEST AND RESULTS

The programs were benchmarked on an HP Pavilion laptop with the following parameters:

| | |
|---|---|
| **Processor** | AMD RYZEN 7 5800H with RADEON GRAPHICS (16 CPUs) ~3.2 Ghz |
| **Memory** | 16 GB |
| **OS** | Windows 10 home 64-bit |

*Table 2. Computer Specifications*

The testing procedure consists of the following, first the cells between A4:T1000 are filled with the function calls, this choice is done since FunCalc supports the range of A1:T1000 and the function logic is written in the cells, thus leaving the range between A1:T3 for this purpose. Then the method from (Anne Benedicte Abildgaard Ejsing, 2021) for conducting benchmarks is applied. It starts by conducting a random sample with the size of 100 benchmark results, then the Cochran's formula is used to find the number of samples needed to determine the final sample size denoted as n in the equation bellow and then n number of benchmarks are run and an average is presented as a final result. The benchmark used is "Full Recalculation"

$$n = \left(\frac{Z_{a/2} * \sigma}{r * \mu}\right)^2$$

*Equation 1. Cochran's formula*

Here Z represents a Z table function and a represents the significance level, σ is the standard deviation is desired error margin and μ is the mean of the population. The significance level is related to the confidence level in the following manner:

$$confidence\ level = 1 - a$$

*Equation 2. Relation between significance and confidence levels*

Meaning that at a 95% confidence level, the significance level would be 0.05, the Z-table function provides a standard Z-score for the desired level of confidence, for

example for a 95% confidence level the score is 1.96 (Anne Benedicte Abildgaard Ejsing, 2021).

First, the factorial function was calculated with the use of the following code for Excel:

```
=LAMBDA(N,IF(N<=0, N, N*Factorial(N-1)))
```

*Listing 1. Code for factorial in Excel*

The following code doesn't use tail recursion for the factorial function. The benchmarking for the factorial function was done on the 10th element (Factorial(10)).

For FunCalc the following code was used, and the testing was done in the same manner as in Excel, FunCalc supports a built-in benchmark technique for FullRecalculation, so no calls to VBA were done:

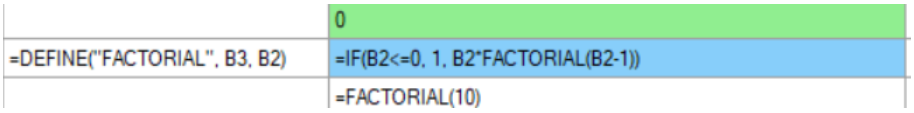


*Figure 9. Code for Factorial in FunCalc*

Another function tested was the Tail-Recursive and Non-Tail Recursive version of the Nth-Fibonacci number, where N was 10 in the case. Other functions were Sum Columns of an array, index of a number of an array, and minimum index

The following results were gotten:

| | **Excel** | | **Funcalc** | |
|---|---|---|---|---|
| **Program** | Number of runs | Runtime | Number of runs | Runtime |
| **Non-Tail Recursive Factorial (10)** | 4655 | 24.02 ms | 756 | 8.9 ms |
| **Non-Tail Recursive Fibonacci(10)** | 3467 | 318.7 ms | 438 | 102.4 ms |

| | | | | |
|---|---|---|---|---|
| **Tail Recursive Fibonacci (10)** | 16775 | 29,5 ms | 475 | 21.1 ms |
| **SumColumns(arr)** | 1884 | 36,4 ms | 5271 | 40.2 ms |
| **INDEXOF(900,arr)** | 9189 | 12.3 ms | 7342 | 5.3 ms |
| **MINDEX(arr)** | 1840 | 80,36 ms | 1550 | 29,3 ms |
| **Groundwater_daily** | 1101 | 1 276.11 ms | 4 | 25033.7 ms |
| **Groundwater_daily without UDF** | 62 | 1 261.29 ms | 52 | 31314.7 ms |
| ***arr is array equal to {500,1000,900,100,230,150,90,20,30,420,6969,1312,14}** | | | | |

*Table 3. Benchmark Results*

In 5 of the cases, FunCalc performed faster than Excel. FunCalc had issues with the Groundwater_daily sheet and the results gotten there seemed suspiciously weird but they were verified with two different scripts.

## 2.5. DISCUSSION OF THE RESULTS

Although FunCalc is faster in most situations, the study results are inconclusive. There are several reasons that this might be, one of the issues for example is that some things in both languages can be called incomparable. For example, one reason could be that FunCalc makes difference between Vertical and Horizontal arrays while Excel does not, logically this can mean that they have different data representations which can affect the runtime. An experiment has been done that compares the runtime of "VARRAY". "HARRAY" (the single-celled representations of the horizontal and vertical in FunCalc) against the arrays in Excel which are multi-celled, the experiment consisted of doing a benchmark of 20,000 copies of an array of the numbers between 1 and 10. In Funcalc the arrays were pasted in the fields between A1:T1000, while in excel the array was pasted from A1:A20000 (the first element of the array, considering that the array is multi-celled), FunCalc also by default "renders" the cells between A1:T1000 and at least the version used in the experiments doesn't allow inserting more rows or columns, though the option exists it just clears the existing row/column. The experiment resulted in similar results in both Excel and FunCalc so the theory of different representations was dismissed.

| Technology name | Result |
|---|---|
| **Excel** | 7,13 ms |
| **FunCalc** | 7,5 ms |

*Table 4. Benchmark of FunCalc and Excel Arrays*

Another theory could be that either FunCalc or Excel has an issue with calculating multi-sheeted books, this theory was also dismissed by an experiment consisting of using a Sheet Defined Factorial function with parameter 10 first in one sheet and then in 10 sheets for both technologies, from this experiment the following results were gotten.

| Excel 1 sheet | Excel 10 sheets | FunCalc 1 sheet | FunCalc 10 sheets |
|---|---|---|---|
| 22,18 ms | 227,25 ms | 9,0 ms | 118,2 ms |

*Table 5. Benchmark of Factorial based on a different number of sheets in each technology.*

The results for one-sheeted workbook with Factorial function are consistent with the results from table 3. This proves that making 10 sheets in both slows down the calculation time by around 10 times for both. So, the theory can be dismissed.

Because of the 6 seconds improvement in Groundwater_daily FunCalc with SDFs but didn't improve the runtime in Excel, another theory was crafted, this theory consists of checking whether the functions are compiled and formulas are interpreted. This experiment consisted of pasting data into the fields between A1:T500 and formulas or functions in the fields A501:T1000, first benchmark a with the formula shown below pasted in the fields and after then the formula is converted to an UDF in both technologies. The aim of this experiment is to prove whether "masking" with UDFs improves the runtime.

```
=AVERAGE($A$1:$T$500)*MIN($A$1:$T$500)
```

*Function 1. Formula used for the masking experiment*

The benchmarks produced the following results, which ultimately proves that "masking" with an SDF doesn't improve the runtime of a program so the 6s improvement in FunCalc can be considered as a deviation.

| Excel | Excel Lambda | FunCalc | FunCalc SDF |
|---|---|---|---|
| **351.351351 ms** | 352.030948 ms | 8685.3 ms | 9431.3 ms |

*Table 6. Results of the experiment that compares Funcalc and Excel with and without SDFs*

This however points out that Excel performs better on "masked" formulas than FunCalc but doesn't perform well on functions that are not a "masking" of existing functions.

## 2.6. CONCLUSION

In conclusion, the results of this part of the study were inconclusive. A reason has not been found why the results of the Groundwater_daily mismatch the other results by a lot, the FunCalc results are however consistent with the results shown in (Møller, 2016). Also using UDFs as "masking" didn't improve the performance dramatically according to the results. In FunCalc a 6-second improvement is noted but the result stays within the same diapason. While most other results are being somewhat consistent in showing that FunCalc is faster. However, Excel performs much better on "masked" formulas and formulas in general than FunCalc while FunCalc proves better on functions that involve recursion. The reason for this is not researched in this project.

## 2.7. FUTURE WORK

For future work, an investigation of the abnormal results of Groundwater_daily one can check if the Excel version from 2022 uses parallel or sequential execution. Another thing that could be done is to do more experiments on large sheets and use more complex functions. Another thing that can be done is to do a more in-depth analysis of the datatype representation in each technology. Another thing that could be done is to investigate why formulas that are "masked" by UDFs are calculated way faster in Excel than in FunCalc.

# CHAPTER 3. ANALYSIS OF COGNITIVE DIMENSIONS

NOTE: Chapter 3 and Chapter 4 are meant to be taken as a whole since they both answer the 2nd question of the problem statement.

In this section, the analysis independent of the user study based on the cognitive dimensions will be done. The analysis will consist of an analysis of some of the dimensions based on comparisons of the languages.

First, the 14 cognitive dimensions will be introduced. The Cognitive Dimensions of Notations are a lightweight approach to analyzing the quality of an existing design or guiding new design decisions. There are 14 cognitive dimensions that can evaluate the design of the programming language.

1. "Abstraction gradient: What are the minimum and maximum levels of abstraction? Can fragments be encapsulated?
2. Closeness of mapping: What 'programming games' need to be learned?
3. Consistency: When some of the language has been learnt, how much of the rest can be inferred?
4. Diffuseness: How many symbols or graphic entities are required to express a meaning?
5. Error-proneness: Does the design of the notation induce 'careless mistakes?
6. Hard mental operations: Are there places where the user needs to resort to fingers or pencilled annotation to keep track of what's happening?
7. Hidden dependencies: Is every dependency overtly indicated in both directions? Is the indication perceptual or only symbolic?
8. Premature commitment: Do programmers have to make decisions before they have the information they need?
9. Progressive evaluation: Can a partially-complete program be executed to obtain feedback on 'How am I doing'?
10. Role-expressiveness: Can the reader see how each component of a program relates to the whole?
11. Secondary notation: Can programmers use layout, colour, other cues to convey extra meaning, above and beyond the 'official' semantics of the language?

12. Viscosity: How much effort is required to perform a single change?
13. Visibility: Is every part of the code simultaneously visible (assuming a large enough display), or it at least possible to juxtapose any two parts side-by-side at will? If the code is dispersed, is it at least possible to know in what order to read it?
14. Juxtapossility: Can different parts of the notation be juxtaposed at the time? ” (M. Petre, 1996)

## 3.1. COGNITIVE DIMENSIONS CHOSEN

Since the delimitations of this study, only a handful of cognitive dimensions will be analyzed by the author. The cognitive dimensions chosen to be analyzed are the ones that the author found more frustrating. The cognitive dimensions chosen are Expressivity, Visibility, and Error-proneness.

### Expressivity

In comparison to Excel, FunCalc can be called long-winded, this is because function definition in FunCalc takes much more space than the one in Excel.

```
=LAMBDA(X,IF(X=0,1,X*Factoriala(X-1)))
```

*Figure 10. Factorial code with Lambda in Excel.*

| B3 | A |
|---|---|
| 1 | 10 |
| 2 | =IF(A1=0, 1, A1*FACTORIALA(A1-1)) |
| 3 | =DEFINE("FACTORIALA", A2, A1) |

*Figure 11. Factorial code in FunCalc*

As you can see the way functions are defined in Funcalc takes much more space. The Factorial function in Excel can be defined in one row, taking fewer characters(38 characters) while in FunCalc the function definition is defined in two cells with a bigger character count(58 without whitespace and not counting the content of A1). So it can be concluded that Excel’s Lambda expressions are far more expressive than FunCalc.

### Visibility

The basic fact that the function definitions in both technologies are hidden in cells on separate sheets or the name manager makes the code not so visible, in FunCalc. multiple function definitions can be viewed but at the same time, it can be argued that it is harder to keep track of the functions scattered in the Function sheet. The fact that the functions take multiple rows and cells means that the programmer has to spend attention on more places when programming in comparison to Excel where the functions are much more compact and are stored inside the "Name Manager".

| A15 | A | B | C |
|---|---|---|---|
| 1 | | | |
| 2 | =A1+B1 | | |
| 3 | =DEFINE("SumTwo", A2, A1, B1) | | |
| 4 | | | |
| 5 | =A4*B4 | | |
| 6 | =DEFINE("MultTwo", A5, A4, B4) | | |
| 7 | | | |
| 8 | =MULTTWO(A7, SUMTWO(B7, C7)) | | |
| 9 | =DEFINE("sumMultiply", A8) | | |

*Figure 12. SDF definitions in FunCalc*

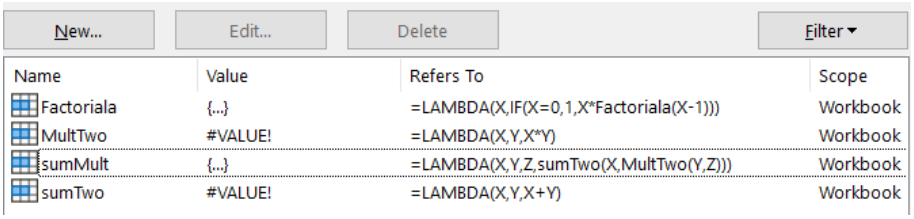


*Figure 13. UDFs definitions in Excel.*

### Error-Proneness

It can be argued that in FunCalc the programmer is more likely to make careless mistakes since the programmer has to keep track of the individual cell addresses and has to spend more time looking at the cells, rather than using a placeholder variable name. Some of the most common mistakes could be:

1. Mistaking the cell addresses when making an expression
2. Mistaking the cell addresses when using the "define" statement

## 3.2. DISCUSSION

It can be argued that Excel provides more intuitive syntax, and better expressivity and keeps the attention of the programmer easily since Excel's syntax is more compact,

doesn’t require memorizing cell addresses when declaring a function, and the programmer will spend more time thinking about the cell addresses when declaring functions instead of working with variables as parameters of the function but it could also be argued that it could be difficult to use the concept of variables for people with no programming backgrounds, this hypothesis can only be proven if the study is extended to people with little to no programming knowledge like people that use spreadsheets only for business things like accounting.

## 3.3. CONCLUSION

In conclusion, based on the analysis of the three cognitive dimensions chosen, Excel offers much more expressive power than FunCalc’s SDFs, also the programmers are less prone to making errors in Excel Lambdas. Both technologies don’t offer a lot of visibility.

# CHAPTER 4. USER TESTING

User testing is conducted by a mixture of Discount Method for Programming Language Evaluation and Cognitive Dimensions of Notations questionnaire. The users are likely not to be familiar with either of the technologies, so a sheet of example programs should be written. According to (Svetomir Kurtev, 2016) the discount method consists of the following procedure:

1. Creating a sheet of sample programs, this sheet should be clear and navigable so the users can browse it easily. The code samples will give the users a better understanding of the language.
2. Estimating the task duration by measuring how fast you can complete the given tasks. Participants are likely to take more time to solve the tasks since they are unfamiliar with the language. It is recommended to have more tasks than the amount the participant is expected to be able to solve but the participant should be aware that it's not expected for him to solve them all.
3. Prepare the setup by choosing the environment that we want the users to use, it can vary from pen and paper to a full coding environment.
4. Gather participants, the golden rule for the number is 5.
5. When the testing is started make sure that the participants are aware that it is not them being tested but the language.
6. Keep the participants talking. Try to talk to the participants and as a facilitator, you might give answers to questions asked by the participants. We are not testing the participants' ability to perform the task but their ability to put it into code in the respective language.
7. After the testing, the participants should be interviewed or given a questionnaire if there are many participants.
8. After all the tests analyze the data and list the problems encountered by the participants. The problems can be split into the following categories:

   Cosmetic problems consist of typos and small keyword and character differences that can be fixed by replacing the wrong part.

   Serious problems consist of structural errors that usually impact code structure, and are small enough to be fixed with a few changes.

   Critical problems consist of fundamental misunderstandings of how the language structures code and large errors of this type would require revision of the code.

A version of the Cognitive Dimensions questionnaire described in (Green, 2000) is being incorporated into the ”interview” part of the Discount method.

## 4.1. STUDY SETUP

The participants for the study are Masters students in the 10th semester of CS-IT programme at Aalborg University. Ideally, the aim for the study size was 5 but due to the time limits, only 3 people participated. The study setup consisted of the participants being seated in front of a laptop with a keyboard and mouse connected to them, all the sessions were screen recorded and the author of the study was a facilitator. The participants were asked to solve the tasks given below both with Excel Lambda expressions and FunCalc. Before solving the tasks they were shown a quick presentation of both. They were shown how to create a simple function that sums up two numbers together and a basic example of recursion. They were also shown how to switch between Function Sheets and Non-Function sheets in FunCalc and how to use the ”Name Manager” in Excel. While solving the tasks the participants communicated with the facilitator and requested help. The participants were also given a cheat sheet of similar programs although neither of the participants felt a need to use it.

The participants had to complete 3 programming tasks that made them use UDFs and one warmup task without UDFs.

> Exercise 1. In a quadratic equation ($a_2 \pm bx \pm c$), having a negative discriminant means that the equation has no solution. Create a function “isSolvable” that calculates the discriminant with given a,b,c as parameters and returns Boolean value “True” if there is solution and ‘False’ if there is not. The discriminant is calculated as follows $D = b_2 - 4ac$. Test it with $2x^2 + 4x - 4 = 0$ i.e. a=2,b=4,c=-4

*Listing 2. First Excercise*

The following task was given since it is easily solvable, contains a conditional, and also makes the participants use basic arithmetic operations.

> Exercise 2. In Absurdia you pay for utilities in advance based on estimation, the estimation is done by calculating the average of the sum of previous 9 months consumption divided by 3 (i.e. the average of 3 slices of 3 month consumption measures or simply put as the average of 3 quarters of the year), then the real consumption is balanced after three months and if it is smaller than the estimate you get money in return, otherwise you have to give more money to the issuer. Calculate a function that takes array of the estimate of the 9 months as an array, the real 3-month expenses as array(which should be summed up) and returns how much money should you should get or pay to/from the issuer (negative amount meaning that you consumed more than paid for and a positive amount meaning that you will get money in return).

*Listing 3. Second Excercise*

This task was given so the participants get a grasp of how the “ranges” work in both languages.

> Exercise 3. A factorial is mathematical operation which represents the multiplication of all numbers from 1 to the desired number i.e. 5! = 1*2*3*4*5, Create a recursive factorial function and test it with a Factorial of 10.

*Listing 4. Third Excercise*

This task was done so the participants get a grasp of how recursion works in both languages.

Then the participants have given the aforementioned cognitive dimensions questionnaire which by their choice they solved in written form, without supervision.

## 4.2. STUDY RESULTS

First, the participants answered a few questions about their backgrounds.

Participant 1 answered that they are very familiar with the concept of functional programming, familiar with spreadsheets and that they were proficient in the use of Excel’s formula language. When asked if they used any similar technologies they answered “no”.

Participant 2 answered that they were somewhat familiar with the concept of functional programming and spreadsheets, they stated that they were unfamiliar with

Excel's formula language. When asked if they've used similar technologies they replied with Python, MATLAB, Julia, and OpenOficeCalc. And that they use the system for data collection.

Participant 3 stated that they were somewhat familiar with functional programming, very familiar with spreadsheets, and proficient in Excel's formula language. Other similar systems they've used were DAX and Python and they use the system for data analysis.

The participants were mostly dissatisfied with the GUI and the bugs in both software and some of them felt demotivated to do the study, especially in FunCalc. All of them successfully completed the study and their feedback is reported below.

Participant 1. During the coding session participant 1 rushed the tasks, he didn't test all the functions. Participant 1 had difficulties with the commas and parentheses in both languages and also a problem with using the "define" function in FunCalc, the participant was confused with cell names used as variable names in FunCalc when defining a function but adapted to it quickly. The participant seemed to get used to the language fast and didn't use principles like TDD. The participant quickly adapted to both FunCalc and Excel and was quick to ask questions to the facilitator.

Participant 2. Participant 2 had done everything in a systematic way, he made use of principles like TDD, and the participant wrote everything in a systematic way. A problem that the participant had was the ranges/lists, the participants found it counterintuitive that when creating a new function that takes a list, the parameter isn't specified to be a list, especially in FunCalc where lists as a parameter are described as a variable/cell name, e.g. SUM(A1) but when calling the function a range is given. An example is shown below:

```
=SUM(A1)
=DEFINE("SUME", A2, A1)
=SUME(B2:B5)
```

*Figure 14. The correct way of defining functions that take a range as an argument in FunCalc*

The participant wondered how to denote the range as a function parameter in FunCalc and one of them tried to denote it in the fashion shown below while doing the 2nd task:

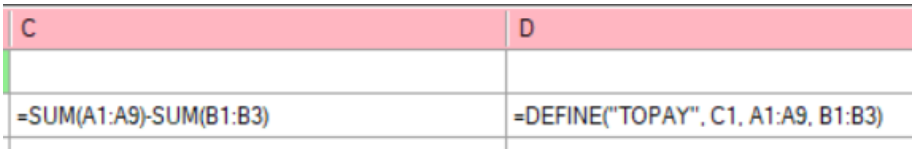


*Figure 15. Mistake done by participant 2*

Participant 3. The last participant wasn't systematic as the first two, the participant quickly finished the tasks, and the participant seemed a little bit careless since he had done the survey after work. The participant shared the general confusion about the FunCalc using cells instead of variables. The participant also had a problem sharing with Participant 1 with the "define" keyword.

| **Participant** | **Cosmetic problems** | **Serious problems** | **Critical problems** |
|---|---|---|---|
| **Participant 1** | Missing parenthesizes and comas in Excel | | |
| **Participant 2** | | | Not being able to grasp how the array structure works in FunCalc. |
| **Participant 3** | Using inappropriate names for the Variables in Funcalc. | Missing the "Define" Keyword when defining the function in FunCalc. Missing the "Lambda" keyword in Excel. | |

*Table 7. List of mistakes done by each participant*

Now the results of the cognitive dimensions questionnaire will be presented.

## Visibility and Juxtaposibility

Participant 1 answered that one issue with the visibility is the comas and parentheses, Participant 2 also said that the language was “parentheses soup” but this was stated under the questions about Viscosity.

Participant 2 also argued that hiding the functions in a different tab wasn’t offering much visibility but this comment was mostly directed to the GUI.

Participant 3 didn’t offer much to the questionnaire.

## Viscosity

Participant 1 found Excel easier to make changes than Funcalc.

Participant 2 had issues with FunCalc keystrokes randomly deleting and pasting content into cells. They also criticized both languages to be difficult to make changes in.

Participant 3 argued that it is hard to keep track of the cell names in FunCalc especially when trying to make a change.

## Diffuseness

Participant 1 didn’t raise any issues with the length of the notation but raised the issue with “naming the variables with cell identifiers”.

Participant 2 stated that both languages were quite expressive but stated also that in FunCalc “function definitions occupied several cells in an unstructured manner that made me want a new sheet for every function definition”

Participant 3 didn’t elaborate anything but stated that both notations were quite brief.

## Hard Mental operations

Participant 1 argued that it was hard to think sequentially when developing a solution and was hard to go back and define a variable, also the participant stated that in their head it was difficult to keep track of variable names in FunCalc since they were represented with cell identifiers.

Participant 2 argued that it was difficult to understand the ranges in FunCalc and Excel but during the coding session they had more issues in FunCalc. Also that the Expression in Excel lambda came directly after the arguments when defining a

function. Also stated that ranges were quite hard to grasp and that in FunCalc "I could not for the life of me understand how they are handled"

Participant 3 found using the "define" keyword in FunCalc pretty hard to grasp, especially the 2nd argument given to it (the pointer to the formula).

## Error proneness

Participant 1 had an issue with keeping track of commas and parentheses, especially when editing the signature of the function.

Participant 2 raised an issue with error reporting in both languages in connection with the syntax errors.

Participant 3 stated that missing parentheses and misspelling the function name was a big issue.

## Closeness of Mapping

Participant 1 stated that it is strange to use the cell identifiers as variables when declaring a function.

Participant 2 said that Excel's notation was closer to the result it was describing and that "Having to refer to cell names in FunCalc removed a lot of the mathy notation". Also, they stated it was quite strange the way that a function that takes a range in FunCalc was declared.

Participant 3 stated only that it was strange to them to use the pointer in the "define" keyword in FunCalc (the 2nd argument).

## Role Expressiveness

Participant 1 stated that the recursion in FunCalc is particularly hard to interpret.

Participant 2 stated that in Excel's notation it was pretty straightforward to tell each part of it when reading it while FunCalc required "several clicks" and "switching between sheets". FunCalc's array handling and argument passing were difficult to interpret, also they stated that error reporting in both languages was "almost non-existent".

Participant 3 said that the notations were well structured.

## Hidden dependencies

Participants agreed that there were no significant hidden dependencies.

## Progressive evaluation.

Participant 1 stated it was quite difficult to stop in the middle of creating a function since it is “very difficult to keep track of long definitions ” and that they can’t see how much progress they’ve made.

Participant 2 stated that it was easy to get a partial result is easy but it is hard to keep track of how much progress you’ve made since viewing code and functions at the same time is impossible.

Participant 3 said that it is easy to stop in the middle of creating a function but only if you properly close the parentheses.

## Provisionality

Participant 1 answered that neither of the languages made it easy to sketch things out when playing with ideas.

Participant 2 stated it was possible to define sample arguments and a function call before defining the function itself.

Participant 3 didn’t answer anything in this section.

## Premature commitment

Participant 1 stated that the system didn’t force them to think in any particular order. And that the only thinking needed to be done in advance is the definition of the “necessary blocks and encapsulating the logic”

Participant 2 stated that writing a function returning a partial result is possible and then refining the calculation itself to return the desired result.

Participant 3 stated the order didn’t matter as well and that one could start by doing either the expression or the definition of the function in FunCalc

## Consistency

Participant 2 answered that the conditional is similar to a function call when asked “Where there are different parts of the notation that mean similar things, is the similarity clear from the way they appear?” and that there are no things that ought to be similar that are made different by the notation.

Participants 1 and 3 didn't answer the questions in this sphere or stated they were "irrelevant" to the type of questions they answered in the coding session.

### Secondary notation

Participant 2 answered that it was possible to make notes to yourself by writing in the cells and that it "is a very nice benefit of sheet-based programming languages".

Participants 1 and 3 didn't use comments during the coding session so they left the section blank.

### Abstraction mechanism

Participant 2 stated that the system doesn't system insist that they start by defining new terms before they can do anything else and stated that although an error is thrown the cell value is saved and that they "love reactivity like that". They also stated that they can use cells in defining a "sort of" variables.

The other participants didn't give significant answers to this section and rather gave neutral answers.

When asked what they could improve the participants stated the following things:

1. "in funcalc a more designated and readable spot for adding the functions, in excel it is quite intuitive and readable"
2. "blend the formula tab (function sheet) and the regular tab (non-function sheet)" (FunCalc)
3. " Allow expressions directly in a FunCalc "DEFINE" function rather than referencing a cell", "Easy multi-line cells/Excel LAMBDA fields", "No difference between function sheets and calculation sheets in FunCalc", and "Fix the FunCalc editor"

## 4.3. DISCUSSION

Most of the participants are dissatisfied with FunCalc, most of the comments they made are connected to using the "define" keyword, using cell identifiers instead of variables, the segregation between Function and non-function sheets, and recursion. The participants preferred Excel's Lambda compared to FunCalc but they also gave criticism to it as well. A lot of participants were confused by the interface of both

technologies and were dissatisfied with the bugs in FunCalc. The participants criticized the Lambdas for not making a special place for the expression but simply putting it after the arguments. The participants also found common flaws in both languages with the use of commas and parentheses some even calling them “parentheses soup”.

## 4.4. CONCLUSION

Based on the answers to this survey and the coding session, it can be concluded that people familiar with computer science prefer Excel’s Lambdas over FunCalc. They found both technologies unfamiliar and struggled to understand them but they ended up preferring Excel. A full copy of the answers to the questionnaire is available in the appendix section of this report.

## 4.5. FUTURE WORK

A further expansion of this study can be done by finding more subjects of different backgrounds, this could diversify the results of the study and conclude if the results of this study are biased on the participant’s background.

# CHAPTER 5. COMPUTATIONAL THINKING

Computational thinking as a concept describes the ability to think like a computer scientist. It is best described as the ability to conceptualize problems like a computer scientist not just program them, computational thinking does not teach one how to think like a computer but how to think using the concepts and limitations of computing, it combines mathematical and engineering thinking, it can be taught to anyone and everywhere, it can be taught to biologists, medical specialists and many others (Wing, 2006).

To answer the 3rd question from the problem statement a rather philosophical study will be done, taking into account the aforementioned user study.

According to (Alfred Aho, 2022) the core idea of computational thinking is abstraction and all abstractions in computer science have two properties.

1. A data model is 1 or more types of data plus the possible relationships between them, for example, the cells in spreadsheets can be represented as graph nodes and the connection between them can be represented as an edge.
2. A way of manipulating that data could be either the Formula language, the Lambda expressions, or the FunCalc's SDF.

Spreadsheets themselves can be used in a way similar to one of the databases, in their basic sense they contain data, and relationships between data (if any). The relationships can be described through formulas, every formula is dependent on the cells given as parameters. The database offers an abstraction of the data by encapsulating the different structures of data records in tables with fields and offers a way to access this data by commands. In comparison, some could argue that spreadsheets create such abstractions by encapsulating the data in a cell and then creating the dependency graph of the cells.

In the study (Sanford, 2018) the author suggests that using spreadsheets is a good way to introduce computational thinking to students and states that the spreadsheets are a good way to introduce computational thinking, the author argues that the spreadsheets are as "visual as pen and paper", and "students can produce useful material with minimal instruction" and that they have "extensive library of features and functions", "graphical presentation is easily produced" and "it's unlikely that they are suppressed anytime soon". The author provided examples of how spreadsheets could be used to solve some mathematical problems. The author concludes that spreadsheets are the medium but there are other options for teaching computational thinking.

It can be also argued that introducing UDFs to the spreadsheets can teach non-programmers some programming aspects such as recursion and Turing completeness. For example, the introduction of Lambdas to Excel can teach people about Lambda Calculus and Turing Machines and since Lambda calculus is equivalent to Turing machine this means that the concept of Turing machine can be explained through Lambda calculus and this is crucial knowledge for computer scientists. The UDFs also offer abstraction overexpression which corresponds with Tennent's principle of abstraction. These factors can contribute to teaching computational thinking to non-computer scientists.

But on the other hand, it can be argued that neither FunCalc nor Excel's lambdas introduce enough abstraction of the data. Neither of those technologies offers information hiding and it could be argued that the data in the cells is rather raw and the cell addresses are not a form of abstraction but rather pointers to the raw data's location. There are only raw data stored in the cell's address and some sort of "range" or "array" structures. There are no abstract data types, objects, or structs.

There is also no language that is used to modify the data itself similar to the one used in databases, there is no way to modify a value of a cell by using UDFs or the Formula Language. Spreadsheets only do computations, but they don't allow the data to be modified by formulas. This could be argued to be the missing key feature that makes spreadsheets suitable for teaching computational thinking. Most programming languages use concepts such as variable assignments (C, C++, Java, etc.). Not having a way to assign/mutate value to cells by code but instead a way to do computations on it.

Some think that these features are crucial to learning for most computer scientists but pure functional programming doesn't allow the assigning and mutating of variables either and both languages are in the functional paradigm, similar to Haskell. Haskell is a purely functional language that supports similar features. There are also a lot of impure programming languages such as Python which is widely used by data scientists too. UDFs support also key components that are included in most programming languages such as conditionals and recursion. Most languages support constructs such as loops and these loops can be presented through conditionals and recursion. One of the types of abstraction is functions which divide the functionality into general-purpose entities that structure the program.

As stated in (Niemelä, 2018) there are some analogies between functional programming and mathematics. The problem-solving technique in both consists of decomposition, solving subproblems, and evaluation of the result. The problem-solving in algebra is similar to the abstraction, automation, and analysis in CT. First the problem-solving starts with abstracting, then after this, the problem is decomposed into smaller solvables and then the result is analyzed.

The spreadsheets themselves are useful for teaching computational thinking already and introducing the UDFs can teach some crucial key points of computer science such as recursion and Lambda calculus (in the case of Excel) but as shown in the user study the introduction of UDFs comes with a lot of drawbacks. Even people with knowledge of Computer Science were dissatisfied with them and found major usability issues in both of the technologies.

Most of the participants of the study found major usability issues in both. Even though they were experienced programmers. Some of them were dissatisfied with the Lambda syntax, others were dissatisfied with the way of defining functions in FunCalc. Some of the participants called both languages “parenthesis soup”. Many of the participants criticized also the way of defining the functions and the “places” where the function definitions were located.

In Excel, the participants gave their criticism for the use of the Name Manager, the parentheses and commas usage, and using the place of the “last argument” as a place where the expression is written.

The participants criticized FunCalc much more than Excel. In FunCalc they criticized the use of function sheets, using cell addresses instead of variable names, and that function definitions take multiple rows.

In the analysis of the cognitive dimensions chapter, it is argued that Funcalc is less expressive in comparison to Excel, also that the programmer is much more prone to serious mistakes when coding in FunCalc. Also, both of the technologies don’t offer much visibility.

Another issue with the UDFs is their use cases, are there suitable use cases of UDFs for people with no or limited understanding of Computational Thinking. Will the people with no computational thinking e.g. accountants have to encounter a situation where UDFs are a must or at least helpful? This question can be answered by analyzing the applications of UDFs and the work activities of the spreadsheet users unfamiliar with CT.

Because of the hardships that experienced programmers had with both technologies, it can be concluded that while UDFs (especially in Excel) can be used to teach crucial concepts of Computer Science, they can’t be recommended as an educational tool.

## 5.1. DISCUSSION AND FUTURE WORK

The focus on this chapter was limited and most of the arguments presented are rather philosophical, since the time and resource limitations and not being able to find subjects from non-Computer Science related backgrounds. One way to expand this part of the study is to put those philosophies to a test by finding test subjects, another

way to expand them is by looking at specific use cases of UDFs, in order to do this analysis of specific use cases should be done, similar to the one in (Sanford, 2018).

# CHAPTER 6. GENERAL CONCLUSIONS OF THE STUDY

In conclusion, all three questions from the problem statement were answered, therefore the study can be considered a successful one.

Q: Which of the methods has better performance?

This question was answered in Chapter 2, the answer to it is that the results of the benchmark experiments were inconclusive. FunCalc was faster in 5 out of 7 of the cases, one of the cases where FunCalc lost to Excel was the Groundwater_daily workbook. Several theories on why it happened have been constructed and all of them were disproved. It should also be noted that introducing UDFs to this workbook as "masks" didn't improve the workbook runtime in FunCalc, however, it should also be noted that Excel performs much better on 'masked' functions and formulas, while FunCalc performs better on functions involving recursion. The reason for this has not been investigated in the project.

Q: Which of the methods is more familiar and understandable to the users?

This question was answered in Chapters 3&4, to answer this question first analysis of three cognitive dimensions was done. The cognitive dimensions chosen for this analysis were Visibility, Expressivity, and Error-Proneness. The analysis found that Excel Lambdas are more expressive than FunCalc's SDFs, programming in Excel Lambdas is less error-prone, and both technologies don't offer good visibility.
In Chapter 4 the results of the user study are explained, the participants preferred Excel's Lambdas over FunCalc's SDFs but were dissatisfied with both.

Q: Can UDFs languages be used to facilitate computational thinking?

To answer this question only a philosophical study was done. While Functional programming has some parallels with mathematics and UDFs can also teach non-programmers concepts like recursion, Turing-completeness, and Lambda Calculus (Excel), the UDFs in spreadsheets can't be recommended for teaching computational thinking since the major usability issues experienced by the participants of the user study.

# CHAPTER 7. BIBLIOGRAPHY

Alfred Aho, J. U., 2022. *Abstractions, Their Algorithms, and Their Compilers.* [Online]
Available at: https://cacm.acm.org/magazines/2022/2/258231-abstractions-their-algorithms-and-their-compilers/fulltext#R9

Anne Benedicte Abildgaard Ejsing, J. R. N. C. S. N. L. R. M. J., 2021. *The Influence of Programming Paradigms on Energy Consumption,* s.l.: s.n.

Anon., u.d. *Excel performance: Improving calculation performance.* [Online]
Available at: https://docs.microsoft.com/en-us/office/vba/excel/concepts/excel-performance/excel-improving-calculation-performance

Bock, A. A., 2019. *A Comparison Between SISAL 1.2 and Funcalc,* s.l.: s.n.

Chase, C., 2020. *SPREADSHEETS ARE OBSOLETE IN THE AGE OF BIG DATA—WHAT IS REPLACING THEM?.* [Online]
Available at: https://demand-planning.com/2020/09/14/spreadsheets-are-obsolete-in-the-age-of-big-data-what-is-replacing-them/

Georgi Zhivkov, K. F. J. P. L. C. W., 2021. *Puffin, A spreadsheet programming language,* s.l.: s.n.

Gislason, H., 2018. *Excel vs. Google Sheets usage — nature and numbers.* [Online]
Available at: https://medium.grid.is/excel-vs-google-sheets-usage-nature-and-numbers-9dfa5d1cadbd

Green, A. F. B. a. T. R., 2000. *A Cognitive Dimensions Questionnaire Optimised for Users.* s.l., s.n.

Gross, C., 2021. *Announcing LAMBDA Helper Functions: Lambdas as arguments and more.* [Online]
Available at: https://techcommunity.microsoft.com/t5/excel-blog/announcing-lambda-helper-functions-lambdas-as-arguments-and-more/ba-p/2576648

M. Petre, T. G., 1996. Usability Analysis of Visual Programming Environments : A 'Cognitive Dimensions' Framework. *Journal of Visual Languages and Computing* .

Møller, N. K., 2016. *Pre-Analyses Dependency Scheduling with,* s.l.: s.n.

Niemelä, P., 2018. *From Legos and Logos to Lambda A Hypothetical Learning Trajectory for Computational Thinking,* s.l.: Tampere University of Technology.

Sanford, J., 2018. *Spreadsheets, Introducing Computational Thinking Through,* s.l.: Springer International Publishing AG, part of Springer Nature 2018.

Sestoft, P., 2017. *Corecalc and Funcalc.* [Online]
Available at: http://www.itu.dk/~sestoft/funcalc/

Simon Peyton Jones, A. B. B., 2003. A User-Centred Approach to Functions in Excel. *ACM SIGPLAN Notices.*

Svetomir Kurtev, T. A. C. B. T., 2016. *Discount Method for Programming Language Evaluation.* s.l., PLATEAU 2016.

Wing, J. M., 2006. *Computational Thinking,* s.l.: Communications of the ACM.

# APPENDICES

# Appendix A. Answers to the survey

Participant 1 answers.

Survey:

Answer the following questions (35 min):

Visibility and Juxtaposability

• How easy is it to see or find the various parts of the notation while it is being created or changed? Why?

It is somewhat easy for both languages

• What kind of things are more difficult to see or find?

In excel commas and parentheses , in funcalc also the variable names are confusing

• If you need to compare or combine different parts, can you see them at the same time? If not, why not?

The scope and visibility of functions are difficult to view at the same time so rather not

Viscosity

• When you need to make changes to previous work, how easy is it to make the change? Why?

It is difficult to find the definitions of functions in funcalc, while in excel it is quite easy as it is organised where the definitions are exactly.

• Are there particular changes that are more difficult or especially difficult to make? Which ones?

In funcalc it is difficult to find the function definition itself, in excel there are not very difficult things to do

Diffuseness

• Does the notation a) let you say what you want reasonably briefly, or b) is it long-winded? Why?

For both languages the notation is very concise, yet for funcalc it is a bit confusing naming the variables with cell identifiers

• What sorts of things take more space to describe?

none

Hard Mental Operations

• What kind of things require the most mental effort with this notation?

It is difficult to think sequentially when developing the solution as when doing sequential operations it is not very intuitive where to go back and define the variable (for both languages)

• Do some things seem especially complex or difficult to work out in your head (e.g. when combining several things)? What are they?

In excel no, in funcal the variable names being represented by cell identifiers

Error Proneness

• Do some kinds of mistake seem particularly common or easy to make? Which ones?

in both languages commas are difficult to keep track of, also parentheses

• Do you often find yourself making small slips that irritate you or make you feel stupid? What are some examples?

When editting the signature of the functions losing track of the syntax more particularly paranthases

Closeness of Mapping

• How closely related is the notation to the result that you are describing? Why?

---

• Which parts seem to be a particularly strange way of doing or describing something?

In funcalc it is strange that cell identifiers are used as variables, in excel things seems simple and logical

Role Expressiveness

• When reading the notation, is it easy to tell what each part is for in the overall scheme? Why?

Yes, after defining the identifiers it is easy to use them

• Are there some parts that are particularly difficult to interpret? Which ones?

In excel no, in funcalc when doing recursion it is weird that using the cell identifier and then think of it as variable and not the cell

• Are there parts that you really don't know what they mean, but you put them in just because it's always been that way? What are they?

no

Hidden Dependencies

• If the structure of the product means some parts are closely related to other parts, and changes to one may affect the other, are those dependencies visible? What kind of dependencies are hidden?

no, none

• In what ways can it get worse when you are creating a particularly large description?

In longer expressions it is difficult in both languages to understand the expressions

• Do these dependencies stay the same, or are there some actions that cause them to get frozen? If so, what are they?

the dependencies stay the same

Progressive Evaluation

• How easy is it to stop in the middle of creating some notation, and check your work so far? Can you do this any time you like? If not, why not?

it is very difficult to keep track of long definitions so it is very difficult

• Can you find out how much progress you have made, or check what stage in your work you are up to? If not, why not?

no, the whole definition will have to be gone through to identify how far the definition is towards completion

• Can you try out partially-completed versions of the product? If not, why not?

irrelevant question

Provisionality

• Is it possible to sketch things out when you are playing around with ideas, or when you aren’t sure which way to proceed? What features of the notation help you to do this?

I didn’t get the feel of either language making it easy to do

• What sort of things can you do when you don’t want to be too precise about the exact result you are trying to get?

Not sure, was not included in the questions set

Premature Commitment

• When you are working with the notation, can you go about the job in any order you like, or does the system force you to think ahead and make certain decisions first?

In both languages the order of definitions is not relevant

• If so, what decisions do you need to make in advance? What sort of problems can this cause in your work?

define necessary blocks that encapsulate the logic

Consistency

• Where there are different parts of the notation that mean similar things, is the similarity clear from the way they appear? Please give examples.

was not covered in exercises

• Are there places where some things ought to be similar, but the notation makes them different? What are they?

no

Secondary Notation

• Is it possible to make notes to yourself, or express information that is not really recognised as part of the notation?

not sure

• If it was printed on a piece of paper that you could annotate or scribble on, what would you write or draw?

write

Abstraction Management

• Does the system give you any way of defining new facilities or terms within the notation, so that you can extend it to describe new things or to express your ideas more clearly or succinctly? What are they?

Yes, we can define multiple functions

• Does the system insist that you start by defining new terms before you can do anything else? What sort of things?

no

• Do you find yourself using this notation in ways that are unusual, or ways that the designer might not have intended? If so, what are some examples?

no

• After completing this questionnaire, can you think of obvious ways that the design of the system could be improved? What are they? Could it be improved specifically for your own requirements?

in funcalc a more designated and readable spot for adding the functions, in excel it is quite intuitive and readable

Participant 2 answers.

Survey:

Answer the following questions both for FunCalc and for Excel: e.g. In Excel that and in FunCalc that

Visibility and Juxtaposability

• How easy is it to see or find the various parts of the notation while it is being created or changed? Why?

For excel, I still do not know to this day where the documentation to the various functions is found. I was, however, able to recall the SUM function from having used it previously.

For Funcalc, I was able to guess the name of the SUM function since it matched the one used in Excel.

• What kind of things are more difficult to see or find?

There was absolutely no chance of using either without documentation. In the GUI, there was no indication on how to solve the task using the tools available.

I will add that in Excel, the "named entities," or whatever, dialog was strangely named for what it does.

• If you need to compare or combine different parts, can you see them at the same time? If not, why not?

It drove me up the wall how the contents of the Funcalc cells would be hidden behind various error messages while not in focus. Having the expression hidden by an error message by default in the function sheets made it harder for me to use.

The fact that Excel did not allow viewing the functions at all (outside their weird dialog) was arguably worse.

Both languages, being sheet-based, were exceptionally good at showing intermediate results and the input variables, since they are all visible in the cells.

Viscosity

• When you need to make changes to previous work, how easy is it to make the change? Why?

In Excel it was merely tedious. Going back to continue working on a formula after checking the intermediate result was mostly pain-free. Though it was weird to have to go into a dialog to do it.

The extreme usability issues with FunCalc's interface meant that making changes became difficult. These issues prevent me from evaluating the language itself, because unexpected deletions of cells, and insertions of characters, as well as inexplicable errors, prevented me from reliably making changes to my code.

A hypothetical well-functioning editor might have been slightly easier to use than the also janky excel counterpart, but having two different kinds of sheets felt backwards. And counter to the sheet-oriented paradigm.

• Are there particular changes that are more difficult or especially difficult to make? Which ones?

Both languages had issues with parenthesis soup, where keeping track of the appropriate number of parentheses was difficult by itself. Finding the appropriate place to make changes even more so, not aided by the interface of either program.

Diffuseness

• Does the notation a) let you say what you want reasonably briefly, or b) is it long-winded? Why?

Expressions in both languages were compact almost to a fault. Function definitions in Excel was also well-hidden and therefore arguably compact. In FunCalc, function definitions occupied several cells in an unstructured manner that made me want a new sheet for every function definition.

• What sorts of things take more space to describe?

In FunCalc, having more arguments to a function takes significantly more space.

Hard Mental Operations

• What kind of things require the most mental effort with this notation?

The ultra-compact conditionals were hard for me to keep track of. Of course this is the case for both languages.

The last argument in the Excel LAMBDA definition function seemed weird to me, but I adapted quickly.

Managing arguments in FunCalc was extremely difficult and not something I ever got comfortable with.

• Do some things seem especially complex or difficult to work out in your head (e.g. when combining several things)? What are they?

Lists were difficult to grasp in both languages. In Excel I got it eventually, in FunCalc I could not for the life of me understand how they are handled.

Error Proneness

• Do some kinds of mistake seem particularly common or easy to make? Which ones?

Both languages had extremely bad reporting of syntax errors.

• Do you often find yourself making small slips that irritate you or make you feel stupid? What are some examples?

Only in interfacing with the UI. Both systems made unexpected and unwanted changes to my input, or handled key strokes differently from what I'd expect.

Closeness of Mapping

• How closely related is the notation to the result that you are describing? Why?

Excel was much closer, since it allowed named arguments. Having to refer to cell names in FunCalc removed a lot of the mathy notation.

• Which parts seem to be a particularly strange way of doing or describing something?

Arrays in FunCalc.

Role Expressiveness

• When reading the notation, is it easy to tell what each part is for in the overall scheme? Why?

In Excel it actually started to be pretty straightforward, despite the small editing window and the parenthesis soup.

Reading a function in Funcalc takes several clicks for switching sheets and reading the obscured values of cells.

• Are there some parts that are particularly difficult to interpret? Which ones?

FunCalc array handling. FunCalc argument passing to an expression, since there is no highlighting of a cell reference and *corresponding* variable.

Error reporting on syntax mistakes is nonexistent for both languages, so I suppose that's pretty hard to interpret.

• Are there parts that you really don't know what they mean, but you put them in just because it's always been that way? What are they?

Array handling in Funcalc. Argument passing in FunCalc.

Hidden Dependencies

• If the structure of the product means some parts are closely related to other parts, and changes to one may affect the other, are those dependencies visible? What kind of dependencies are hidden?

Both being functional languages, there are not many hidden dependencies.

The actual expression of a function (its definition as well) is hidden in FunCalc until you click into it.

• In what ways can it get worse when you are creating a particularly large description?

Both languages are designed for having one-liners, which is just terrible.

Possibly you can calculate your expression in more  than one cell in FunCalc? But the UI was so janky that I did not fancy trying to experiment with that.

• Do these dependencies stay the same, or are there some actions that cause them to get frozen? If so, what are they?

Well FunCalc just fully became unusable due to something being stuck. It is unclear if it was a bug or a hidden dependency.

Progressive Evaluation

• How easy is it to stop in the middle of creating some notation, and check your work so far? Can you do this any time you like? If not, why not?

It required a few clicks for both, but it was very easy by just returning a partial result.

• Can you find out how much progress you have made, or check what stage in your work you are up to? If not, why not?

This is hard because for both, looking at the code while looking at a result at the same time is not possible.

• Can you try out partially-completed versions of the product? If not, why not?

Aye.

Provisionality

• Is it possible to sketch things out when you are playing around with ideas, or when you aren't sure which way to proceed? What features of the notation help you to do this?

Yes, it was possible for me to define sample arguments and a function call, before actually defining the function. In both languages. This helped me think about what I wanted the function to do and what its signatureshould be.

• What sort of things can you do when you don't want to be too precise about the exact result you are trying to get?

I have no idea.

Premature Commitment

• When you are working with the notation, can you go about the job in any order you like, or does the system force you to think ahead and make certain decisions first?

By returning some partial result, I can start with any part of the calculation and then refine it how I like. However, once I've written in one part of the calculation, I sort of have to keep it in my function and add to it, rather than make several small components.

• If so, what decisions do you need to make in advance? What sort of problems can this cause in your work?

I did not think about my decisions that much. These were small programs.

Consistency

• Where there are different parts of the notation that mean similar things, is the similarity clear from the way they appear? Please give examples.

I suppose the conditional is strikingly similar to a function call. The definition notation as well.

• Are there places where some things ought to be similar, but the notation makes them different? What are they?

No, can't say there is.

Secondary Notation

• Is it possible to make notes to yourself, or express information that is not really recognised as part of the notation?

Yes, I added the variable names above all the sample arguments. This is a very nice benefit of sheet-based programming languages.

• If it was printed on a piece of paper that you could annotate or scribble on, what would you write or draw?

I might actually be able to write out a sample cell structure.

Abstraction Management

• Does the system give you any way of defining new facilities or terms within the notation, so that you can extend it to describe new things or to express your ideas more clearly or succinctly? What are they?

I can sort of do variables (pi for example) by hard-coding them into certain cells. I'm sure it is also possible to do function calls to simpler functions. I suppose I did that with the SUM function. Though writing my own function to call from another never came up.

• Does the system insist that you start by defining new terms before you can do anything else? What sort of things?

Actually not. It throws an error, but saves the cell value. I love reactivity like that.

I'm not sure FunCalc always accepted my error-cells, but I think so.

• Do you find yourself using this notation in ways that are unusual, or ways that the designer might not have intended? If so, what are some examples?

I don't believe that I did.

• After completing this questionnaire, can you think of obvious ways that the design of the system could be improved? What are they? Could it be improved specifically for your own requirements?

Fix the FunCalc editor.

Allow expressions directly in a FunCalc “DEFINE” function rather than referencing a cell.

Easy multi-line cells/Excel LAMBDA fields.

No difference between function sheets and calculation sheets in FunCalc.

Participant 3 answers.

Survey:
Answer the following questions (35 min):
Visibility and Juxtaposability
• How easy is it to see or find the various parts of the notation while it is being created or changed? Why?
It is easy
• What kind of things are more difficult to see or find?
_____
_____
_____
_____
_____
_____
• If you need to compare or combine different parts, can you see them at the same time? If not, why not?
You can
Viscosity
• When you need to make changes to previous work, how easy is it to make the change? Why?
it's easy to make the change, you just go the the formula tab

• Are there particular changes that are more difficult or especially difficult to make? Which ones?

maybe the variables that they have to be [letter][number] format

Diffuseness

• Does the notation a) let you say what you want reasonably briefly, or b) is it long-winded? Why?

a

• What sorts of things take more space to describe?

-

Hard Mental Operations

• What kind of things require the most mental effort with this notation?

having to point to the formula's code and having to write the define key word every time

• Do some things seem especially complex or difficult to work out in your head (e.g. when combining several things)? What are they?

-

Error Proneness

• Do some kinds of mistake seem particularly common or easy to make? Which ones?

miss parantesis, misspel the function name

• Do you often find yourself making small slips that irritate you or make you feel stupid? What are some examples?

-

Closeness of Mapping

• How closely related is the notation to the result that you are describing? Why?

Very close

• Which parts seem to be a particularly strange way of doing or describing something?

pointing to the functions code

Role Expressiveness

• When reading the notation, is it easy to tell what each part is for in the overall scheme? Why?

yeah, it's well structured

• Are there some parts that are particularly difficult to interpret? Which ones?

_____
_____
_____
_____
_____
_____

• Are there parts that you really don't know what they mean, but you put them in just because it's always been that way? What are they?

_____
_____
_____
_____
_____
_____

Hidden Dependencies

• If the structure of the product means some parts are closely related to other parts, and changes to one may affect the other, are those dependencies visible? What kind of dependencies are hidden?

_____
_____
_____
_____
_____
_____

• In what ways can it get worse when you are creating a particularly large description?

haven't experienced that one

• Do these dependencies stay the same, or are there some actions that cause them to get frozen? If so, what are they?

_____
_____
_____
_____
_____
_____

Progressive Evaluation

• How easy is it to stop in the middle of creating some notation, and check your work so far? Can you do this any time you like? If not, why not?

you can do this at any time but you have to properly end the formula

• Can you find out how much progress you have made, or check what stage in your work you are up to? If not, why not?

yes

• Can you try out partially-completed versions of the product? If not, why not?

_____
_____
_____
_____
_____
_____

Provisionality

• Is it possible to sketch things out when you are playing around with ideas, or when you aren’t sure which way to proceed? What features of the notation help you to do this?

i don't know if it has this feature

• What sort of things can you do when you don’t want to be too precise about the exact result you are trying to get?

_____
_____
_____
_____
_____
_____

Premature Commitment

• When you are working with the notation, can you go about the job in any order you like, or does the system force you to think ahead and make certain decisions first?

I think you can do it in any of the 2 directions write function's code first and then the fucntion definition or vice versa

• If so, what decisions do you need to make in advance? What sort of problems can this cause in your work?

_____
_____
_____
_____
_____
_____

Consistency

• Where there are different parts of the notation that mean similar things, is the similarity clear from the way they appear? Please give examples.

_____
_____
_____
_____
_____
_____

• Are there places where some things ought to be similar, but the notation makes them different? What are they?

no

Secondary Notation

• Is it possible to make notes to yourself, or express information that is not really recognised as part of the notation?

i don't know

• If it was printed on a piece of paper that you could annotate or scribble on, what would you write or draw?

describe the function's definition elements

Abstraction Management

• Does the system give you any way of defining new facilities or terms within the notation, so that you can extend it to describe new things or to express your ideas more clearly or succinctly? What are they?

new functions

• Does the system insist that you start by defining new terms before you can do anything else? What sort of things?

no

• Do you find yourself using this notation in ways that are unusual, or ways that the designer might not have intended? If so, what are some examples?

_____
_____
_____
_____
_____
_____

• After completing this questionnaire, can you think of obvious ways that the design of the system could be improved? What are they? Could it be improved specifically for your own requirements?

blend the formula tab and the regular tab together

# Appendix B. User tasks and Questions asked before.

Survey

You have 1 hour to solve this questionnaire, in the 2nd section you will encounter some programming tasks which you will have 30 mins to solve,

the tasks are not used to evaluate your programming skills but to evaluate the language that you program in.
First you will watch a video (5 mins)
1st. Something about yourself(5 min)
How familiar are you with the concept of Functional Programming?

☐1-Not familiar ☐2 –Somewhat familiar ☐3 – Familiar ☐4- Very familiar

☐5- Everyday use

How familiar are you with Spreadsheets?

☐1-Not familiar ☐2 –Somewhat familiar ☐3 – Familiar ☐4- Very familiar

☐5- Everyday use

Do you consider yourself proficient in Formula language use?

☐1-Yes ☐2 –No

Have you used other similar systems? I.e. other data analysis languages like python, R, Matlab (If so, please name them)

_____
_____
_____
_____
_____

What task or activity do you use the system for?

_____
_____
_____

What is the end-product of using the system?

_____
_____
_____

Programming Tasks: Write in both FunCalc and Excel the following programs. You are not obliged to write all the programs but as much as the time permits, you are allowed to use the cheat sheet..
Exercise 0. Without defining a function calculate the volume of a sphere. Defined as $V = 4/3\ \pi r^3$.
Exercise 1. In a quadratic equation ($ax_2 \pm bx \pm c$), having a negative discriminant means that the equation has no solution. Create a function “isSolvable” that calculates the discriminant with given a,b,c as

parameters and returns Boolean value “True” if there is solution and ‘False’ if there is not. The discriminant is calculated as follows $D = b_2 - 4ac$. Test it with $2x_2 + 4x - 4 = 0$ i.e. a=2,b=4,c=-4.

Exercise 2. In Absurdia you pay for utilities in advance based on estimation, the estimation is done by calculating the average of the sum of previous 9 months consumption divided by 3 (i.e. the average of 3 slices of 3 month consumption measures or simply put as the average of 3 quarters of the year), then the real consumption is balanced after three months and if it is smaller than the estimate you get money in return, otherwise you have to give more money to the issuer. Calculate a function that takes array of the estimate of the 9 months as an array, the real 3-month expenses as array(which should be summed up) and returns how much money should you should get or pay to/from the issuer (negative amount meaning that you consumed more than paid for and a positive amount meaning that you will get money in return).

Exercise 3. A factorial is mathematical operation which represents the multiplication of all numbers from 1 to the desired number i.e. 5! = 1*2*3*4*5, Create a recursive factorial function and test it with a Factorial of 10.

# Appendix C. Code used for doing a Full Recalculation in Excel.

```
Private Declare PtrSafe Function getFrequency Lib "kernel32" _

Alias "QueryPerformanceFrequency" (cyFrequency As Currency) As Long

Private Declare PtrSafe Function getTickCount Lib "kernel32" _

Alias "QueryPerformanceCounter" (cyTickCount As Currency) As Long

'

Function MicroTimer() As Double
```

```
'


' Returns seconds.
'
    Dim cyTicks1 As Currency
    Static cyFrequency As Currency
    '
    MicroTimer = 0

' Get frequency.
    If cyFrequency = 0 Then getFrequency cyFrequency

' Get ticks.
    getTickCount cyTicks1

' Seconds
    If cyFrequency Then MicroTimer = cyTicks1 / cyFrequency
End Function


Sub FullcalcTimer()
    DoCalcTimer
```

```
End Sub


Sub DoCalcTimer()
    Dim dTime As Double
    Dim dOvhd As Double
    Dim oRng As Range
    Dim oCell As Range
    Dim oArrRange As Range
    Dim sCalcType As String
    Dim lCalcSave As Long
    Dim bIterSave As Boolean
    Dim Stringify As String
    Dim inte As Integer
    Dim avg As Double
    Dim popmean As Double
    Dim stdev As Double
    Dim numb As Double
    Dim arr(100) As Double
    Dim zerocount As Integer
    Dim opa As Double
    On Error GoTo Errhandl
```

```
' Initialize
    dTime = MicroTimer

    ' Save calculation settings.
    lCalcSave = Application.Calculation
    bIterSave = Application.Iteration
    If Application.Calculation <> xlCalculationManual Then
        Application.Calculation = xlCalculationManual
    End If
    sCalcType = "Full Calculate open workbooks: "
    avg = 0
    For inte = 0 To 100
' Get start time.
        dTime = MicroTimer
        Application.CalculateFull

' Calc duration.
        dTime = MicroTimer - dTime
        On Error GoTo 0

        dTime = Round(dTime, 5)
        arr(inte) = dTime
```

```
        avg = avg + dTime

    Next inte
     popmean = avg / 100
     avg = 0
    For inte = 0 To 100
        avg = avg + (arr(inte) - popmean) ^ 2
    Next inte
     stdev = Math.Sqr(avg / 100)
     numb = ((1.96 * stdev) / (0.005 * popmean)) ^ 2

     MsgBox sCalcType & " " & CStr(numb) & " times", _
        vbOKOnly + vbInformation, "CalcTimer"
     avg = 0
    For inte = 0 To CInt(numb)

        dTime = MicroTimer
        Application.CalculateFull

' Calc duration.
        dTime = MicroTimer - dTime
        On Error GoTo 0
```

```
        dTime = Round(dTime, 5)
        avg = avg + dTime
        If dTime > 0.01 Then countzeroes = countzeroes + 1
    Next inte
    opa = avg / CInt(numb)

    MsgBox sCalcType & " " & CStr(avg) & " Seconds", _
        vbOKOnly + vbInformation, "CalcTimer"
Finish:

    ' Restore calculation settings.
    If Application.Calculation <> lCalcSave Then
        Application.Calculation = lCalcSave
    End If
    If Application.Iteration <> bIterSave Then
        Application.Calculation = bIterSave
    End If
    Exit Sub
Errhandl:
    On Error GoTo 0
    MsgBox "Unable to Calculate " & sCalcType, _
```

```
        vbOKOnly + vbCritical, "CalcTimer"

    GoTo Finish

End Sub
```

NOTE: Due to software bug the division avg/numb is done manually.

# Appendix D. Cochran's formula in FunCalc source code.

```
private void BenchmarkWorkbook(WorkbookForm wf, int runs,
string benchmarkName, Func<long> benchmark) {
      Log("=== Benchmark workbook called: ");
      List<double> parts = new List<double>();
      List<double> finalresult = new List<double>();
      Stopwatch stopwatch = new Stopwatch();
      double z = 1.96;
      double stdev = 0;
      double popMean=0;
      for (int i = 0; i < runs; i++) {
        stopwatch.Reset();
        stopwatch.Start();
        benchmark();
        stopwatch.Stop();
        double average = stopwatch.ElapsedMilliseconds;
        parts.Add(average);

        Log(String.Format("[{0}] Average of the {1} runs:
{2:N2} ms",
                          benchmarkName, runs, average));
        wf.SetStatusLine((long)(average + 0.5));
            }
```

```
        popMean = parts.Average();
        stdev = Math.Sqrt(parts.Average(v => Math.Pow(v -
popMean, 2)));
        double n = Math.Pow((1.96*stdev)/(0.005*popMean), 2);
      Log(String.Format("[{0}] is N",
                            (int) n));
            double  rex = 0;
            int countzeroes = 0;
            if(n>0) {
            for (int i = 0; i < (int) n; i++)
            {
                stopwatch.Reset();
                stopwatch.Start();
                benchmark();
                stopwatch.Stop();
                double average =
stopwatch.ElapsedMilliseconds;
                rex +=  average;
                if(average>0.01)
                    countzeroes++;



            }
            Log(String.Format("[{0:0.0}] is result",
                           rex/(double) n) );
            }
           else
            {
                Log(String.Format("undefined N"
                           ));
            }
        }
```

# Appendix E. CHEATSHEET

# Excel programs

Money after interest are calculated by the formula:
A = P(1 + rt). Where P is the initial amount of money invested, r is the interest rate and t is the time.Write a function that calculates the monetary gain after interest rate.

```
=LAMBDA(p,r,t,p*(1+r*t))
```

Eligible for discount. In Munchausen shops, you are eligible for discount if you are under 18 or above 65, and earning less than 40,000 kronnor. Make a function that checks if you are eligible for discount.

```
=LAMBDA(age,income,IF(OR(age<18,AND(age>65,income<40000)),TRUE,FALSE))
```

Tax exemption. In Absurdia tax exemption is calculated based on the sum of the income for each month divided by four(4). If this results in a result lower than 50,000 then the person is exempt from taxation. Create a function that takes array of incomes as input and returns “true” if the result is 50,000 else returns “false”.

```
=LAMBDA(x,IF(SUM(x)>50000,TRUE
,FALSE))
```

The Fibonacci sequence is a sequence of numbers that Write a function that returns the N-th Fibonacci number.

```
=LAMBDA(N,IF(N<=1,0,NFib(N-
1)+NFib(N-2)))
```

# Funcalc programs

Money after interest are calculated by the formula:

A = P(1 + rt). Where P is the initial amount of money invested, r is the interest rate and t is the time.Write a function that calculates the monetary gain after interest rate.

| C2 | A | B | C |
|---|---|---|---|
| 1 | | | |
| 2 | =A1*(1+B1*C1) | | |
| 3 | =DEFINE("interest", A2, A1, B1, C1) | | |

Eligible for discount. In Munchausen shops, you are eligible for discount if you are under 18 or above 65, and earning less than 40,000 kronnor. Make a function that checks if you are eligible for discount.

| B2 | A | B |
|---|---|---|
| 1 | | |
| 2 | =IF(OR(A1<18, AND(A1>65, B1<40000)), 1, 0) | |
| 3 | =DEFINE("discount", A2, A1, B1) | |

Tax exemption. In Absurdia tax exemption is calculated based on the sum of the income for each month divided by four(4). If this results in a result lower than 50,000 then the person is exempt from taxation. Create a function that takes array of incomes as input and returns “true” if the result is 50,000 else returns “false”.

| A6 | A |
|---|---|
| 1 | |
| 2 | =IF(SUM(A1)/4<50000, 1, 0) |
| 3 | =DEFINE("taxexemption", A2, A1) |

The Fibonacci sequence is a sequence of numbers that Write a function that returns the N-th Fibonacci number.

| A4 | A |
|---|---|
| 1 | |
| 2 | =IF(A1<=1, 0, NFIB(A1-1)+NFIB(A1-2)) |
| 3 | =DEFINE("NFIB", A2, A1) |

# Appendix F. Dictionary and explanation of some terms

UDF= User-defined functions, this includes both SDFs and Lambdas

SDFs= Subcase of UDF, the way UDFs are defined in FunCalc

Excel Lambdas/ Excel UDF= Subcase of UDF, the way UDFs are defined in Excel.