ACTA DIDACTICA NAPOCENSIA
Volume 9, Number 1, 2016

INTRODUCTION TO CLASSROOM SPREGO

Mária Csernoch, Piroska Biró

Abstract: Sprego is programming with spreadsheet functions. The present paper provides introductory Sprego examples which have so far only been available in Hungarian. Spreadsheet environments offer both a programming tool which best serves beginner and end-user programmers' interest, and an approach which lightens the burden of coding and language details. Sprego programming utilizes these tools and offers a real world problem solving method based on authentic tables with various contents. The examples selected for the paper show how Sprego programming can be introduced into programming and/or spreadsheet classes.

Key words: Sprego programming, functional language, concept based problem solving, computational thinking

1. Introduction

Sprego was officially introduced as a high-mathability [1]–[3], deep metacognitive computer problem solving approach [4] for teaching spreadsheets, in Hungary in 2014 [5]. Several papers, in both English and Hungarian, have been published describing the essence of the method, its advantages compared to the "classical", "user-friendly" approaches, its effectiveness, and various sample tasks and their solutions [6]–[11]. However, the most essential, introductory classroom examples and methods are only available in Hungarian [5]. We have found that there is a pressing need to make this content available worldwide. Teachers do not feel comfortable with the Hungarian coursebook, and beyond that, those who do not have access to the book find the papers with the more advanced and expanded problems quite complicated. Furthermore, we have experienced that some teachers find it difficult to leave behind the methods which they are familiar with, and apply a completely new approach without any introductory guidelines.

2. The state of art

2.1. The black-sheep of the digital world

It is interesting that spreadsheet management – end-user computing in general – is not accepted as something "serious" by computer science. "[End-user computing] seems to be invisible to the central corporate IT group, to general corporate management, and to information systems (IS) researchers." [12]. End-user computing is treated badly, based on misconceptions, as illustrated by the following citations. "Instead of children bored out of their minds being taught how to use Word and Excel by bored teachers, …" [13]. "Computer science … is not just a collection of low-level routine knowledge such as how to format pages in a word processor, or how to make tables in HTML:" [14].

The digital world requires a high number of professional programmers and highly qualified end-users, but there are not enough of them. It is also likely that at our present rate of development this lack of competent professionals will increase.

2.2. Introducing Sprego

Sprego is an abbreviation for Spreadsheet Lego [5], [8], [9]. The name originated from the idea that – similar to Lego –, it applies basic, general purpose building blocks to solve real world spreadsheet problems based on authentic tables. The name is borrowed from Lego; however, the method is from

Received April 2016.

"hard-code" programming. In Sprego, while relying heavily on the functional language of spreadsheets, we do programming. As has been proved in previously published works [15]–[17], one of the advantages of teaching introductory programming in functional languages is the simplicity of the languages and the fact that students are familiar with the concept of function. Consequently, the focus is not on the details of the language and on the coding, but on the concept and algorithm of the problems. The advantages of the approach, however, extend further, since it is not only effective in introductory courses, but also in end-user computing. With Sprego we can teach programming to the wider public, to those participants of the digital world who do not want to do "hard-code" programming, even with educational purpose software (EPS) [18, 19].

Sprego serves this purpose unconditionally in an environment with which both students and end-users are familiar. In addition, spreadsheets have become one of the most widely accepted software tools in the "real world", so it is not only educational purpose software which is involved. Consequently, spreadsheets are also accepted by those end-users who do not feel like programming [20, 21].

Sprego fulfills both of the requirements of the digital world. It considers end-users who are capable of programming and provides a method which serves as an introductory programming course for future programmers and as a target language for end-users. The approach also works as a tool for developing students' computational thinking, which is a requirement of the digital era in the near future [22].

3. Sprego

3.1. Sprego functions

The basic building blocks of Sprego are the set of Sprego functions [5]. We claim that a dozen general purpose functions can serve as the basic set for programming. These are the Sprego12 functions. This number of functions is in accordance with the predicted number of instructions users are able to handle in programming [23] and the average number of functions used in spreadsheets [24].

Table 1. *Sprego12 functions*

Sprego12 – Sprego-dozen	
Sprego1	Sprego2
SUM()	MATCH()
AVERAGE()	INDEX()
MIN()	ISERROR()
MAX()	
LEFT()	
RIGHT()	
LEN()	
SEARCH()	
IF()	

However, we must emphasize that the set of Sprego functions is an open one, similar to the Lego set, so additional general purpose functions can be accepted and added, according to the requirements of the particular problem.

The original Sprego functions set contains the groups Sprego1 and Sprego2, composing Sprego12, a dozen functions, altogether. Sprego2 requires a higher level of algorithmic skill, and the handling of vectors and errors, which explain their semi-separation from Sprego1.

In our experience, the functions listed in Sprego3 have proved to be the most common addition to Sprego12. Sprego3 functions are an extension of some of the Sprego12 functions (MIN()—SMALL(), MAX()—LARGE()), handling vectors, matrixes, and the indexes of their items (ROW(), COLUMN(),

OFFSET(), TRANSPOSE()), logical functions from hard-core programming (AND(), OR(), NOT()), solving maths problems (RAND(), INT(), ROUND()), and a text based function, SUBSTITUTE(), to handle the language differences and spelling errors and mismatches.

3.2. Composite functions

The Sprego functions serve as one of the essential elements of the approach. To solve "serious" real world problems, these functions have to be built into composite functions, which is the second tool of Sprego [5]. To demonstrate this building process we heavily rely on the Matrjoska-doll concept, and illustrate the method used to create embedded formulas and composite functions with these dolls. For the time being, with the help of our pre-service teachers of informatics, we have created a set of Sprego-dolls and demo programs to encourage better understanding of the Sprego concept [25].

3.3. Array formulas

The third Sprego tool is the array-approach [24, 26]. It has been proved that one of the main sources of spreadsheet errors originates in the copying of formulas [27, 28]. With array formulas copying can be avoided, so spreadsheets would become less erroneous.

In imperative languages, the handling of variables, and especially arrays, are demanding tasks. In spreadsheet environments, with the aid of the graphical interface, the definition of the array is a simple selection and the evaluation of the formula defined on an array is similar to loops for or foreach.

There are two different array formulas in spreadsheets: single-result array formulas (SR-AF) and array-result array formulas (AR-AF). Both types of array formulas should be evaluated with the Ctrl + Shift + Enter key combination [26, 24], [5]–[11].

With a single-result array formula, the formula is created in a cell where at least one of the arguments overwrites the default one-value argument and should be evaluated as an array formula.

With an array-result array formula, the method is expanded with the definition of the array. The steps taken are the following: (1) Create the array formula in the first cell of the array (evaluating with Ctrl + Shift + Enter), where the output of the first item is displayed (all items are calculated, but only the first item is displayed, since the array is not yet defined). (2) Define the array, which is a selection starting from the cell of the formula to the last cell involved (at present, the white selection cross in Excel, not the black autofill). (3) Go back to the source code (use the F2 function key, or click on the editing bar, or double click on the cell of the formula) and then re-evaluate the formula with Ctrl + Shift + Enter. All the items of the array are displayed.

3.4. Further advantages of Sprego

Without giving further details, which are available in previous publications [5]–[11], we will only illustrate the essence of these advantages.

Sprego relies heavily on the building of concepts and algorithms of the original problems, the debugging and discussing of the results, and the generalization of the problems. This approach is similar to hard-core [29] and educational programming [23], [30] and to Pólya's approach to problem solving [31], and is in accordance with the level of mastery in computer science curricula [32]. According to this problem solving approach, we do real programming in a soft-wave environment: initialized as "real-programming" with "soft-coding" (RP-SC).

Only authentic sources, tables are used, to convince students that spreadsheets are useful tools, and not "only" EPS, and to make the transfer between the school and the real world smoother.

Using general purpose, basic functions is software independent, due to the simplicity of the Sprego functions. Consequently, the method can be used in any MS Excel version and also in Open and LibreOffice Calc.

Functional languages rely heavily on the concept of function, mainly adapted from maths studies. On the other hand, the Sprego approach, with its numerous tasks and examples to work with, strengthens the development of this concept, and also the concept of composite and n-ary functions, and of the n-dimensional vector. These subjects are considered high-level mathematics, and until now have barely been mentioned in primary and secondary education. This is an opportunity which was never available before the appearance of these programming tools.

3.5. Curricula compatibility

The nature of Sprego brings one further advantage. It focuses on developing the students' computational thinking, which should be one of the basic skills of the digital era [22] and which would revolutionize the teaching of maths [33, 34] and ICT. Consequently, Sprego suits general ICT curricula where both application and programming studies are taught. With this approach both requirements can be fulfilled with the least effort using only one language: a simple language, a familiar, user-friendly interface, and one language to cover two subfields of the ICT curricula [35]. What is more, Sprego offers connections to, and field practice for, mathematics, and through its authentic contents, a strong connection with the various school subjects and real world problems.

However, we must emphasize at this point that teachers are first of all required to make these revolutionary changes in education. It has been proved in natural sciences that teachers who themselves believe in the "incremental" nature of science and have high teaching self-efficacy are the ones who can lead their students to efficient real world problem solving [36]. With this approach we aim to clear up the misconception that programming and computer problem solving starts with turning on the computer. We argue that real coding should be preceded by design [37], and that this approach is not the privilege of hard-code programming [14].

4. Introductory Sprego examples

The following problems are intended to show how an authentic table serves as a data source, how real world problems can be composed, and their solutions designed and coded. The coding examples provide details of how composite functions and array formulas can be constructed based on the table and the problems at issue. The focus is on the design and on the concept of the problems, with Sprego providing the simplified language for coding.

The *movies* table (Figures 1 and 2) is downloaded from the IMDB website [38], which is a popular page, frequently visited by school children. Considering the content of this table, students are quite familiar with it and would be greatly interested in data retrieval.

As mentioned in the previous sections, the Sprego approach follows Pólya's concept based problem solving method [31]. In the following examples, we start with the data analyses to see what the table offers. In the next step the problem is presented in a natural language sentence. This problem has to be analyzed, translated into algorithm and then into code. The final step is the assessment, which is one of the most crucial steps in the process. Assessment includes "…considering the concept from a multiple viewpoints and/or justify the selection of a particular approach to solve a problem. It implies more than using a concept; it involves the ability to select an appropriate approach from understood alternatives." [32].

4.1. Movies table

The original table has four columns: Rank, Rating, Title, and Votes, from columns A to D (Figure 1), respectively. However, the Title field holds not only the titles of the movies but also the years in which they came out, two pieces of data which should be separated. The separated years and titles are presented in the modified table in columns E and F (Figure 2).

	A	B	C	D
1	Rank	Rating	Title	Votes
2	1	9.2	The Shawshank Redemption (1994)	790425
3	2	9.2	The Godfather (1972)	585045
4	3	9	The Godfather: Part II (1974)	371343
5	4	8.9	Pulp Fiction (1994)	621174
6	5	8.9	The Good, the Bad and the Ugly (1966)	244920
7	6	8.9	12 Angry Men (1957)	194680
8	7	8.9	Schindler's List (1993)	414133
9	8	8.8	The Dark Knight (2008)	736027
240	239	8	The Searchers (1956)	38369
241	240	8	Ed Wood (1994)	99398
242	241	8	King Kong (1933)	47115
243	242	8	Arsenic and Old Lace (1944)	37338
244	243	8	The Nightmare Before Christmas (1993)	118926
245	244	8	His Girl Friday (1940)	27275
246	245	8	The Philadelphia Story (1940)	34568
247	246	8	Patton (1970)	51797
248	247	8	In the Mood for Love (2000)	36933
249	248	8	The Untouchables (1987)	120523
250	249	8	Papillon (1973)	42632
251	250	8	Castle in the Sky (1986)	38305

Figure 1. *The original IMDB table of the 250 most popular movies*

	A	B	D	E	F
1	Rank	Rating	Votes	Year	Title
2	1	9.2	790425	1994	The Shawshank Redemption
3	2	9.2	585045	1972	The Godfather
4	3	9	371343	1974	The Godfather: Part II
5	4	8.9	621174	1994	Pulp Fiction
6	5	8.9	244920	1966	The Good, the Bad and the Ugly
7	6	8.9	194680	1957	12 Angry Men
8	7	8.9	414133	1993	Schindler's List
9	8	8.8	736027	2008	The Dark Knight
240	239	8	38369	1956	The Searchers
241	240	8	99398	1994	Ed Wood
242	241	8	47115	1933	King Kong
243	242	8	37338	1944	Arsenic and Old Lace
244	243	8	118926	1993	The Nightmare Before Christmas
245	244	8	27275	1940	His Girl Friday
246	245	8	34568	1940	The Philadelphia Story
247	246	8	51797	1970	Patton
248	247	8	36933	2000	In the Mood for Love
249	248	8	120523	1987	The Untouchables
250	249	8	42632	1973	Papillon
251	250	8	38305	1986	Castle in the Sky

Figure 2. *The converted IMDB table*

4.2. Problem 1 – AR-AF

Separate the years from the title-year strings and display them in column E.

Specialties of Problem 1

- The years are on the right side of the original string.
- They are in a pair of parentheses.
- They are numbers.

– They are four digits long.

Algorithm of Problem 1

– Cutting out five characters from the right side of the original string. The output is a five-character-long string. F1.
– Cutting out four characters from the left side of the new string. The output is a four-character-long string. F2.
– Converting the four-character-long string into a four-digit-long number. The output is the years. F3.

Coding of Problem 1

The output of the problem is an array whose items are displayed in column E, in the array of E2:E251 (Figure 3 and Table 2).

In the coding process an array formula (AF) is created from the beginning and expanded step by step, according to the algorithm. F1.

In the first step of the coding the AF is created in E2 (Figure 3A) and evaluated with the CTRL + SHIFT + ENTER key combination. One of the arguments of the RIGHT() function is the array of the original title-year text, an argument which turns the formula into array formula. The output is the first item of the vector (Figure 3A). Following this, the whole vector should be defined (Figure 3B), the source code activated (Figure 3C), and the formula re-evaluated with CTRL + SHIFT + ENTER (Figure 3D). The output is a vector of five-character-long text: years followed by a closing parenthesis.

F1. {=RIGHT(C2:C251,5)}

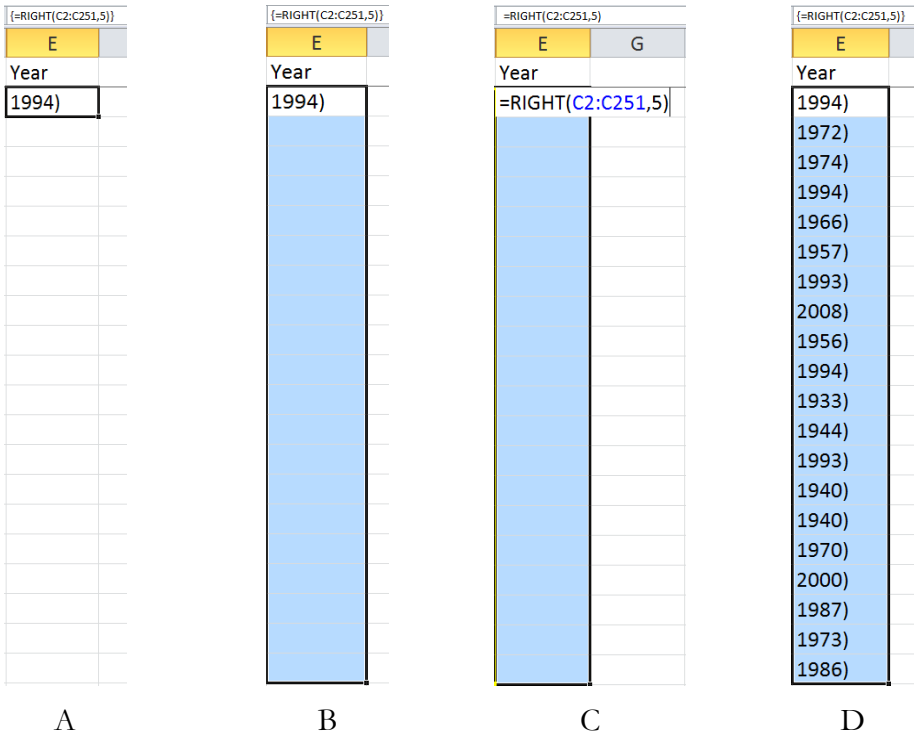

Figure 3. *Thre process of creating array formulas (AR-AF)*

The second step of the algorithm is when the four characters of the number are cut and displayed, leaving the parenthesis behind. The output is a vector of four-character long strings, which look like numbers, but are not. F2.

F2. {=LEFT(RIGHT(C2:C251,5),4)}

In the third step the fake numbers should be converted into numbers. The position of character-digits clearly demonstrates the data type: strings are arranged on the left side of the cells, while numbers on the right side. The conversion is carried out by a multiplication. The fake numbers are multiplied by 1. F3.

F3. {=LEFT(RIGHT(C2:C251,5),4)*1}

Table 2. *Separating the years*

	F1	F2	F3
The Shawshank Redemption (1994)	1994)	1994	1994
The Godfather (1972)	1972)	1972	1972
The Godfather: Part II (1974)	1974)	1974	1974
Pulp Fiction (1994)	1994)	1994	1994
The Good, the Bad and the Ugly (1966)	1966)	1966	1966
12 Angry Men (1957)	1957)	1957	1957
Schindler's List (1993)	1993)	1993	1993
The Dark Knight (2008)	2008)	2008	2008
The Searchers (1956)	1956)	1956	1956
Ed Wood (1994)	1994)	1994	1994
King Kong (1933)	1933)	1933	1933
Papillon (1973)	1973)	1973	1973
Castle in the Sky (1986)	1986)	1986	1986

4.3. Problem 2 – AR-AF

Separate the titles from the title-year strings and display them in column F.

Specialties of Problem 2a

- The titles are on the left side of the original string.
- The lengths of the titles vary.
- The length of the title is 7 characters less than the length of the original string.

Algorithm of Problem 2a

- Deciding the length of the original string. The output is a vector of whole numbers. F4.
- Calculating the length of the title. The output is a vector of whole numbers. F5.
- Cutting out the title from the left side of the original string. The output is the vector of titles. F6.

Coding of Problem 2a (Table 3)

F4. {=LEN(C2:C251)}

F5. {=LEN(C2:C251)-7}

F6. {=LEFT(C2:C251,LEN(C2:C251)-7)}

Table 3. *Separating the titles, using the length of the original string*

	F4	F5	F6
The Shawshank Redemption (1994)	31	24	The Shawshank Redemption
The Godfather (1972)	20	13	The Godfather
The Godfather: Part II (1974)	29	22	The Godfather: Part II
Pulp Fiction (1994)	19	12	Pulp Fiction
The Good, the Bad and the Ugly (1966)	37	30	The Good, the Bad and the Ugly
12 Angry Men (1957)	19	12	12 Angry Men
Schindler's List (1993)	23	16	Schindler's List
The Dark Knight (2008)	22	15	The Dark Knight
The Searchers (1956)	20	13	The Searchers
Ed Wood (1994)	14	7	Ed Wood
King Kong (1933)	16	9	King Kong
Papillon (1973)	15	8	Papillon
Castle in the Sky (1986)	24	17	Castle in the Sky

Assessment of Problem 2

There is another solution to find the length of the titles: the length of the title is two characters less than the position of the opening parenthesis.

Problem 2b – AR-AF

Separate the titles from the title-year strings and display them in column F.

Specialties of Problem 2b

- The titles are on the left side of the original string.
- The lengths of the titles vary.
- The length of the title is 7 characters less than the length of the original string.

Algorithm of Problem 2b

- Deciding the position of the parenthesis. The output is a vector of whole numbers. F7.
- Calculating the length of the title. The output is a vector of whole numbers. F8.
- Cutting out the title from the left side of the original string. The output is the vector of titles. F9.

Coding of Problem 2b (Table 4)

F7. {=SEARCH("(",C2:C251)}

F8. {=SEARCH("(",C2:C251)-2}

F9. {=LEFT(C2:C251,SEARCH("(",C2:C251)-2)}

Table 4. *Separating the titles, using the position of the (character*

	F7	F8	F9
The Shawshank Redemption (1994)	26	24	The Shawshank Redemption
The Godfather (1972)	15	13	The Godfather
The Godfather: Part II (1974)	24	22	The Godfather: Part II
Pulp Fiction (1994)	14	12	Pulp Fiction
The Good, the Bad and the Ugly (1966)	32	30	The Good, the Bad and the Ugly
12 Angry Men (1957)	14	12	12 Angry Men
Schindler's List (1993)	18	16	Schindler's List
The Dark Knight (2008)	17	15	The Dark Knight
The Searchers (1956)	15	13	The Searchers
Ed Wood (1994)	9	7	Ed Wood
King Kong (1933)	11	9	King Kong
Papillon (1973)	10	8	Papillon
Castle in the Sky (1986)	19	17	Castle in the Sky

4.4. Problem 3 – SR-AF

Type a year in cell G2. Calculate the number of movies which came out in G2 year.

Specialties of Problem 3

- The G2 years should be recognized.
- The G2 years should be marked in some way. Since we are planning to find out the number of movies which came out in the G2 year, it is a convenient solution if we mark each match with 1.

Algorithm of Problem 3

- Asking yes/no questions to separate the G2 years from the others. The output is a vector of trues and falses. F10 or F13.
- Deciding on the output if the movie came out in G2 year. In the case of a yes answer the output is 1, in any other cases false. The output is a vector of 1s and falses. F11 or F14.
- Adding the item of the vector. The output is a whole number. F12 or F15.

Solutions A and B use the same algorithm. The only difference between them is that Solution A displays all the items of the vectors and – in a separate cell – the output number, while Solution B uses only one cell, where the first item of the vectors are displayed and finally, the output number.

Coding of Problem 3 – Solution A

Formulas F10 and F11 go into cell H2 and the vector output into H2:H251, and finally this vector serves as the argument of the SUM() function.

F10. {=E2:E251=G2}

F11. {=IF(E2:E251=G2,1)}

F12. =SUM(H2:H251)

Coding of Problem 3 – Solution B

In this solution only one cell is used and expanded through all the steps.

F13. {=E2:E251=G2}

F14. {=IF(E2:E251=G2,1)}

F15. {=SUM(IF(E2:E251=G2,1))}

4.5. Problem 4

Create a dynamic text for Problem 3 which follows the changes in G2: The number of movies which came out in G2.

Specialties and algorithm of Problem 4

- The text has three sections: string1—variable—string2.
- These sections have to be concatenated. F16 or F17

Coding of Problem 4

F16. ="The number of movies which came out in "&G2&"."

Assessment of Problem 4

The output text is more user-friendly if the output number is included in the text. For example: The number of movies which came out in 1994 is 6. In this case the structure of the formula is the following: string1—variable—string2—formula—string3.

F17. ="The number of movies which came out in "&G2&" is "&SUM(IF(E2:E251=G2,1))&"."

4.6. Problem 5

Type a year in cell G2. Calculate the total sum of votes for movies which came out in G2 year.

The specialties and the algorithm of Problems 3 and 5 are the same. The only difference is that while Problem 3 calculates the number of movies from G2 by adding the collected 1s, in Problem 5 we have to collect the votes, which are stored in vector d2:d251, and add them.

F18. {=E2:E251=G2}

F19. {=IF(E2:E251=G2,D2:D251)}

F20. {=SUM(IF(E2:E251=G2,D2:D251))}

4.7. Problem 6

Type a year in cell G2. Calculate the average votes for movies which came out in G2 year.

The specialties and the algorithm of Problems 3, 5, and 6 are the same. The only difference from Problem 5 is that the average of the selected votes has to be calculated, which is coded with an outside AVERAGE() function.

F21. {=E2:E251=G2}

F22. {=IF(E2:E251=G2,D2:D251)}

F23. {=AVERAGE(IF(E2:E251=G2,D2:D251))}

Assessment of Problems 3, 5, and 6

With the same specialties and algorithm, several similar conditional problems can be solved without introducing any problem-specific function. The steps are the following: (1) setting the condition – asking a yes/no question –, (2) deciding on the output values according to the answers, and (3) setting the operation – function or operator – which accepts the selected values from the previous step as argument or operand, respectively.

With the concept of creating composite functions, instead of applying the ever increasing number of built-in, problem specific functions, we have the opportunity to formulate various types of composite conditions with calculated values as operands. We must note here that the database functions of spreadsheets would also serve our purposes; however, spreadsheet-users are not necessarily educated in database management, especially not young school children. Consequently, the database functions cannot be introduced for young children and beginners.

5. Conclusions

We have demonstrated the essence of Sprego, which is a concept and algorithmic approach in spreadsheet environments and introductory programming, where problems solved using authentic tables.

We must emphasize here that programming in Sprego does not require any additional tool, simply a spreadsheet environment, which is usually installed on computers, nor does it need VBA. It is programming in a functional language. Most end-users are familiar with the concept of function, and beyond this knowledge, we rely heavily on the simplicity of the language, which best suits the interests of end-users and beginners. With this approach the focus is not on the coding details but on the concept and algorithm of the problems. Considering these advantages of Sprego, programming with this approach is much less demanding than with imperative languages.

Beyond the theoretical background, the paper provides details of an authentic table – a popular webpage converted to a spreadsheet worksheet – and tasks based on this table which have proved effective in beginner classes.

The problem solving process follows Pólya's approach, which has proved to be effective and efficient both in maths and "hard-core" programming. The steps of this problem solving process are detailed in all the presented tasks and, with the first problem, the coding details, too. It was found earlier that students are better at understanding text based functions than mathematic relations, so the first couple of problems deal with strings, while the second half of the tasks with numbers.

The theoretical background, supported with the examples, clearly show that Sprego requires a change in the thinking mode. Sprego aims to develop schemata, which leads users from attention thinking mode to automated thinking mode [39]–[41]. This change in thinking mode has revealed itself to be less error prone than the widely accepted, classical, user-friendly, but low mathability computer problem solving approaches.

References

[1] Baranyi, P., & Gilányi, A. (2013). Mathability: Emulating and enhancing human mathematical capabilities. In P. Baranyi, A. Esposito, M. Niitsuma & B. Solvang (Eds.) *2013 IEEE 4th International Conference on Cognitive Infocommunications* (CogInfoCom) (pp. 555–558). http://doi.org/10.1109/CogInfoCom.2013.6719309.

[2] Biró, P., & Csernoch, M. (2015a). The mathability of computer problem solving approaches. In *2015 6th IEEE International Conference on Cognitive Infocommunications* (CogInfoCom) (pp. 111–114). http://doi.org/10.1109/CogInfoCom.2015.7390574

[3] Biró, P., & Csernoch, M. (2015b). The mathability of spreadsheet tools. In *2015 6th IEEE International Conference on Cognitive Infocommunications* (CogInfoCom) (pp. 105–110). http://doi.org/10.1109/CogInfoCom.2015.7390573

[4] Csernoch, M., & Biró, P. (2015a). Computer Problem Solving. In Hungarian: Számítógépes problémamegoldás, TMT, *Tudományos és Műszaki Tájékoztatás, Könyvtár- és információtudományi szakfolyóirat*, 62(3), 86–94.

[5] Csernoch, M. (2014). *Programming with Spreadsheet Functions: Sprego*. In Hungarian, Programozás táblázatkezelő függvényekkel – Sprego. Műszaki Könyvkiadó, Budapest.

[6] Csernoch, M. & Balogh, L. (2010). *Algorithms and Spreadsheet-management – Talent Support in Education in the Field of Informatics*. In Hungarian: Algoritmusok és táblázatkezelés. Magyar Tehetségsegítő Szervezetek Szövetsége, Budapest. http://tehetseg.hu/sites/default/files/16_kotet_net_color.pdf, accessed 15-July-2014.

[7] Csernoch, M. (2015). *Algorithms and Schemata in Teaching Informatics*. Debreceni Egyetemi Kiadó, Debrecen. Retrieved January 25, 2016, from http://tanarkepzes.unideb.hu/szaktarnet/kiadvanyok/algoritmusok_es_semak_2.pdf.

[8] Csernoch, M., & Biró, P. (2015b). Sprego Programming. *Spreadsheets in Education (eJSiE),* 8(1). Retrieved from http://epublications.bond.edu.au/ejsie/vol8/iss1/4.

[9] Csernoch, M., & Biró, P. (2015c). *Sprego programming*. LAP Lambert Academic Publishing. ISBN-13: 978-3-659-51689-4.

[10] Csernoch, M., & Biró, P. (2015d). Problem Solving in Sprego. In: Simon Thone, Grenville J. Caroll (ed.) *16th EuSpRIG Annual Conference "Spreadsheet Risk Management"*. London, Five Star Printing Ltd., 1–13.

[11] Biró, P., & Csernoch, M. (2014). An Algorithmic Approach to Spreadsheets. In Hungarian, *Interdiszciplináris pedagógia és a fenntartható fejlődés*. Eds: Buda András, Kiss Endre, DE Neveléstudományok Intézete, Debrecen, 310–321.

[12] Panko, R., & Port, D. (2013). End User Computing: The Dark Matter (and Dark Energy) of Corporate It. *Journal of Organizational and End User Computing, 25*(3), 1–19.

[13] Gove, M. (2012). Digital literacy campaign – Michael Gove's speech in full. BETT 2012. The Guardian. Retrieved January 25, 2016, from http://www.theguardian.com/education/2012/jan/11/digital-literacy-michael-gove-speech

[14] Bell, T. & Newton, H. (2013). *Unplugging Computer Science. Improving Computer Science Education.* (Eds.) Djordje M. Kadijevich, Charoula Angeli, and Carsten Schulte. Routledge.

[15] Booth, S. (1992). *Learning to program: A phenomenographic perspective*. Gothenburg, Sweden: Acta Universitatis Gothoburgensis.

[16] Hubwieser, P. (2004). Functional Modelling in Secondary Schools Using Spreadsheets. *Education and Information Technologies*, 9(2), 175–183. http://doi.org/10.1023/B:EAIT.0000027929.91773.ab

[17] Schneider, M. (2005). A Strategy to Introduce Functional Data Modeling at School Informatics. In R. T. Mittermeir (Ed.), From *Computer Literacy to Informatics Fundamentals* (pp. 130–144). Berlin Heidelberg, Germany: Springer. Retrieved from http://link.springer.com/chapter/10.1007/978-3-540-31958-0_16.

[18] Soloway, E. (1993). Should We Teach Students to Program? *Communications of the ACM, 36*(10), 21–24. http://doi.org/10.1145/163430.164061.

[19] Ben-Ari, M. (2011). Non-myths about programming. Proceeding. ICER '10 *Proceedings of the Sixth international workshop on Computing education research. Communications of the ACM. 54*(7).

[20] Sestoft, P. (2011). Spreadsheet technology. Version 0.12 of 2012-01-31. IT University Technical Report ITU-TR-2011-142. Copenhagen, Denmark: IT University of Copenhagen.

[21] Warren, P. (2004). Learning to Program: Spreadsheets, Scripting and HCI. In *Proceedings of the Sixth Australasian Conference on Computing Education 30* (pp. 327–333). Darlinghurst, Australia: Australian Computer Society, Inc. Retrieved January 25, 2016, from http://dl.acm.org/citation.cfm?id=979968.980012.

[22] Wing, J. M. (2006). Computational Thinking. *Communications of the ACM. (49)*3, 33–35.

[23] Hromkovič, J. (2014). *Introduction to Programming with LOGO.* In German: Einführung in die Programmierung mit LOGO. Wiesbaden, Germany: Springer Fachmedien Wiesbaden.

[24] Walkenbach, J. (2010). *Excel 2010 Bible*. Retrieved December 13, 2015, from http://www.seu.ac.lk/cedpl/student%20download/Excel%202010%20Bible.pdf.

[25] Csapó, G. & Sebestyén, K. (2015). Oktatóprogram a Sprego táblázatkezelő módszerhez. In: Szlávi Péter, Zsakó László (eds.) *INFODIDACT 2015.* Conference, Zamárdi, Magyarország, 1–18. Retrieved December 14, 2015 from http://people.inf.elte.hu/szlavi/InfoDidact15/Manuscripts/CsGSK.pdf

[26] Walkenbach, J. (2002), *Excel 2002 Formulas*. M&T Books, Wiley.

[27] Panko, R. R. (2008). What We Know About Spreadsheet Errors. *Journal of End User Computing's. Special issue on Scaling up End User Development.* 10(2), 15–21.

[28] EuSpRIG Horror Stories (n.d.). Retrieved July 18, 2015, from http://www.eusprig.org/horror-stories.htm.

[29] Backhouse, R. (2011). *Algorithmic Problem Solving*. Padstow, Cornwall, United Kingdom: Wiley.

[30] Hromkovič, J. (2009). *Algorithmic Adventures*. Berlin, Heidelberg, Germany: Springer Berlin Heidelberg.

[31] Pólya, G. (1954). *How to solve it*. A New Aspect of Mathematical Method. (2nd Edition 1957), Princeton, NJ: Princeton University Press.

[32] Computer Science Curricula 2013. Curriculum Guidelines for Undergraduate Degree Programs in Computer Science. December 20, 2013. *The Joint Task Force on Computing Curricula Association for Computing Machinery (ACM) IEEE Computer Society*. http://www.acm.org/education/CS2013-final-report.pdf, accessed 12-April-2014.

[33] Wolfram, C. (2010, July 15). *Stop Teaching Calculating, Start Teaching Math—Fundamentally Reforming the Math Curriculum. Transcript: Wolfram Technology Conference 2010 Talk.* TED Global 2010. Retrieved October 12, 2015, from http://www.computerbasedmath.org/resources/Education_talk_transcript.pdf.

[34] Wolfram, C. (2015). Evidence: Let's promote not stifle innovation in education. Uploaded: May 22, 2015. Retrieved: December 12, 2015. from http://www.conradwolfram.com/home/2015/5/21/role-of-evidence-in-education-innovation.

[35] Csernoch, M. & Biró, P. (2015). Sprego in the ICT Curricula. In Hungarian: Sprego helye az informatika tantervekben. In: Szlávi Péter, Zsakó László (eds.) *INFODIDACT 2015*. Conference, Zamárdi, Magyarország, 1–13. Retrieved January 22, 2016 from http://people.inf.elte.hu/szlavi/InfoDidact15/Manuscripts/CsMBP.pdf.

[36] Chen, J. A., Morris, D. B. & Mansour, N. (2015). *Science Teachers' Beliefs. Perceptions of Efficacy and the Nature of Scientific Knowledge and Knowing*. In International Handbook of Research on Teachers' Beliefs. (Eds.) Fives, H. & Gill, M. G. Routledge.

[37] Angeli, C. (2013). *Teaching Spreadsheets: A TPCK Perspective.* In Improving Computer Science Education. (Eds.) Djordje M. Kadijevich, Charoula Angeli, and Carsten Schulte. Routledge.

[38] IMDB Top Rated Movies. Retrieved January 20, 2013 from http://www.imdb.com/chart/top.

[39] Panko, R. R. (2013). The Cognitive Science of Spreadsheet Errors: Why Thinking is Bad. In *2013 46th Hawaii International Conference on System Sciences* (Vol. 0, pp. 4013–4022). Wailea, HI. http://doi.org/10.1109/HICSS.2013.513.

[40] Panko, R. R. (2015), "What We Don't Know About Spreadsheet Errors. Today: The Facts, Why We Don't Believe Them, and What We Need to Do." Presented at *EuSpRIG 2015*, London, UK, July 9, 2015. Retrieved February 9, 2016 from http://www.eusprig.org/presentations/Presented%20EuSpRIG%202015%20What%20We%20Don't%20Know%20About%20Spreadsheet%20Errors.pdf

[41] Kahneman, D. (2011). *Thinking, Fast and Slow*. New York: Farrar, Straus; Giroux.

Authors

Mária Csernoch, University of Debrecen, Debrecen, Hungary, e-mail: csernoch.maria@inf.unideb.hu

Piroska Biró, University of Debrecen, Debrecen, Hungary, e-mail: biro.piroska@inf.unideb.hu