ACTA DIDACTICA NAPOCENSIA
Volume 9, Number 1, 2016

HOW TO TEACH PROGRAMMING INDIRECTLY – USING SPREADSHEET APPLICATION

Zsuzsanna Szalayné Tahy

Abstract: It is a question in many countries whether ICT and application usage should be taught. There are some problems with IT literacy: users do not understand the concepts of a software, they cannot solve problems, and moreover, using applications gives them more problems. Consequently, using ICT seems to slow work down. Experts suggest learning programming to get some practice in computational thinking; but learning programming is very difficult for many people. In this publication a method is suggested how to teach computational thinking and how to prepare students for programming. The focus is set on the spreadsheet software, but the method is more general: the teaching of programming is embedded into the teaching of application usage. While discussing a specified software or problem to solve, indirectly, one teaches computational thinking and programming. Advanced spreadsheets are known as programming like language but in this case the focus is on the spreadsheets as a program in everyday use. While examining how to solve a problem using spreadsheet and what does spreadsheet do, computational thinking and key competences of programming are developing. Later known data types, data structures, algorithms, functions and methods give the conceptual basis of programming practice.

Keywords: computational thinking, programming, spreadsheet, teaching method

1. Introduction

Using ICT is very easy and it is getting easier and easier every day. There are several comics about the usage of digital tools everywhere and every time and it is getting true in more and more countries. But the low level of digital literacy gives rise to personal, commercial and social problems. The cultural level of a society depends on how and why its members use digital tools. Nowadays the question is how digital literacy can be improved in a society whose members use digital tools every day at home and at work in professional practice [10, 11, 1, 5, 6].

One of the most important key stages is to understand how digital tools work. As they are based on written programs, the starting point of improvement is to teach programming. This is the idea behind national curriculums [13, 12, 4], where programming is set as key competence and, moreover, students have dedicated lessons to programming.

This solution is logical but raises some questions:

- What level of programming knowledge is expected? Is everybody (most of the students) able to reach that level?
- What level of programming knowledge is required to solve problems by using applications?

There are several students who (or whose parents) think ‘programming is too difficult’ but there are others who ‘learn programming on their own’. There are many studies monitoring groups of students being taught programming using new methods[2, 3].These studies conclude that the ‘new method is better than the old one’, but these results are validated locally. Success rather depends on the personality of the teacher and the affinity of the students.

In the last years a new approach has been found, which enables you to teach the basic competencies even to the students who believe learning programming is a ‘mission impossible’. Like the principle of Sysło’s ‘Computational thinking’ [7, 8], the method concentrates rather on the improvement of the mentality than doing explicit code writing. The main point of the approach is an indirect teaching method: students explore software, understand the idea behind the tools and only teachers know they teach basic skills of programming.

Received February 2016.

The mainstream is to teach programming and hope that digital literacy will also improve [15, 16]. In this publication the inverse of this method is introduced: while learning office applications even programming competences will develop. The way of achieving it is very important: one has to look into, to take apart and to understand what and why, to explore how applications work. This approach leads to programming.

The method for improving programming skills is demonstrated through the example of the well-known spreadsheet application. This method is close to problem-type oriented method, defined by Zsakó and Szlávi [9] but in this case the programming knowledge derives from the examination of the used application.

Spreadsheet is a commonly used office application program for organization, analysis and storage of data in tabular form but someone use spreadsheets in special fields like a simple database, form, animation maker or a special programming environment of a functional language [3]. We do not deal with these possibilities. Focusing on the normal usage of spreadsheets we examine the way to improve digital literacy in connection with knowledge and skills of programming.

2. Paradigms of Teaching Using Spreadsheets

Spreadsheet: Software and Application

To learn the menu of spreadsheets application was the main purpose of informatics education. Like in the case of every other software or tools, the basics of spreadsheets software are written in user's documentation. There are several textbooks to help this with screenshots to show where the useful items you have to use are. From the point of view of informatics methodology we have to clear that this way of learning is acceptable by older generation but not effective to learn problem-solving. Moreover, the alpha-generation does not like this type of learning, they prefer short "how to" videos but mostly they like "click & play" practice to explore the menus. It seems students do not need to be taught how to use software.

Spreadsheet is special as it is the mainly used office software to solve problems and to analyze data. Therefore using spreadsheets are taught as a tool which can be used in a wide area of learning and working. Spreadsheet is an application you can use in several cases and the teaching practice focus on how to use the application in typical ones [17, 19, 18]. Typical problems are subject specific therefore the application could be taught in different subjects in many country. Though there are countless different ways to learn how to use spreadsheets – for example to prepare 4th module of ECDL – none of them is about the background of the solutions [14, 20].

The main problem with these teaching practices is that the only question is 'How can I solve this task?' and never asked 'How does spreadsheet work?'. The result is that the development of learners' computational thinking and digital literacy are very poor.

How Does It Work? – Learn Programming!

We could list a lot of initiations to teach how programs work. Nowadays one of the most popular ways is to teach Scratch. Learning programming in Scratch gives you some knowledge about algorithms, functions and data types but this knowledge is not useful in solving problems given by used office applications. It seems that the programs written by students in Scratch and spreadsheets are too far from each other.

Many say that learning Scratch helps to learn other programming languages but in practice, writing program in Scratch is a game for students. They write funny stories without understanding the concepts of programming. There are many concepts hidden by visual programming language which students have to understand later. What is more, the next step, coding – writing codes – gives difficulties as instead of drag&drop accurate typing is expected.

A programmer can solve problems given by spreadsheets. Not because she can code but because she understands the 'hidden concepts'. While learning programming they learn something that enables them

to solve problems in used applications. We have seen, following this learning process, that our student's programming knowledge grows for a while but many of them remain stuck before they reach the proper level. The level which is higher than to the improvement of digital literacy requires but necessary to transfer knowledge between different fields of applications.

Spreadsheet as a Program

Spreadsheet is a software, it is an application but also a program written by programmers. If we teach programming in order to understand how an application works, we should not neglect the given application, in this case the spreadsheet.

In our approach to teach applications and programming are joined. While we are speaking about an application – about a definite spreadsheet – we are using programming terms and talking about programming concepts. Typical questions are: How is it possible? What has happened? What does it mean? Why? What does the program do?

The next four examples are based on MS Excel 2013. Some answers could be different if we choose another spreadsheet application but the concepts are the same. Every example shows a well-known simple task whose solution – normally – takes only 5 minutes but sometimes it gives unsolvable difficulty.

Example 1 – Data Types

"Write your presentation timing in minutes (Hungarian: in 'perc') into a form."

Teaching Excel as a software we say: Type the time and press enter to finish the input. When we teach Excel as application, we talk about strings and numbers, we present number formatting. It takes 10 minutes, learners need more time to get routine in writing and formatting but it is not too difficult. But in practice if someone writes ' p.' instead of 'perc' (Hungarian minute), the result is strange (Figure 1.)

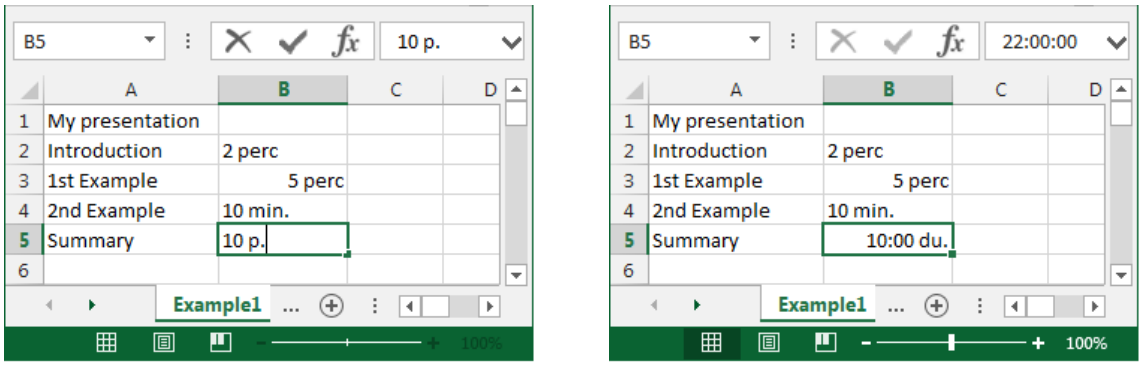

Figure 1. *Unexpected interpretation*

Though writing '15 p' or '9.5 p', the input is correct. "Excel or my spreadsheet seems to be hysterical, poor, dummy…"

Teaching Excel as a program means much more. It starts from typing: what we type is a *string* and pressing enter button the program *parse* the string. Typing 10, Excel gets a string which contains 2 *characters*. Excel 'understands' these characters are digits and, if there is no other character, it *converts* all digits into one number… Excel uses three base data types: string, number and a logical type; you can see using general indent whether Excel has converted the input string into some kind of number (right indented) or logical (centred). Excel 'understands' not only a row of digits but decimal separator, currencies, the % sign and several date and time formats. It can be seen, the value before the percent sign is 100 times of the stored value. It also can be seen that dates are some kind of numbers… or are they?... Excel's parser is a program, working by written rules (algorithms). It cannot read your mind. It

does not understand you think ‘ p.’ means ‘ perc’ but it has a rule: for numbers of [1; 12]∩ℕ it means ‘pm’ and converts the string to a special time format, changing the stored value.

While students learn some rules hidden in Excel, they learn basic concepts of programs, they learn programming. Teaching programming languages we teach how to input and convert data. In this case, students explore how it works in Excel. We have to know, teaching it takes at least 50 minutes instead of 5 minutes but we teach computational thinking, programming concepts and we improve our students’ digital literacy.

Example 2 – Limits of Spreadsheets

“How many cells are there in a spreadsheet? How many characters could be written into one Excel 2013 document?”

To answer the question we can explore Excel as software, we can get practice in scrolling, in paging, in jumping and using go to. Counting the number of columns gives a really math challenge. We can explore Excel as an application, using references (A1: 1; B1: =A1 + 1 and copy…) or automatically numbering to see how many columns are side by side.

Exploring Excel as a program, we have to focus on the fact that sheets have definite number of rows and columns. What does Excel do while we insert a new row or column? Moreover, how many sheets can we insert? How many characters can we write into a cell? Can we evaluate the file size of a full written Excel spreadsheet? What are the defined limits and what are the limits in practice? How can I solve the problem if it goes over the limits? What are the limits in other spreadsheet programs? Is there a program which can process infinite amount of data?

Questions take us away. Inserting data into a limited array involves some knowledge about data structures and algorithms. Counting involves some mathematical knowledge, but exploring the reality of the result gives basic hardware knowledge and we can make a bypass to teach planning rules of programs’ size and complexity.

Example 3 – Cost Accounting, the Equality Problem

“I spent some days in England and during this time I noted how much cash I got and spent. Once I counted my money, it was £19.52. When I left England, I had £8.89 in my pocket. Please, verify whether my accounting is correct.”

This example seems to be right to teach how formulas and functions are given. Practically we teach the difference between addition (+) and summary (=SUM()). We can talk about the number of parameters and clarify there is no subtraction function... The task does not tell us the time or number of rows where £19.52 were in the pocket. This involves to show how we can use the quick sum of the status bar.

This is a very common task which should not cause any problem. We have to get zero as a difference in normal cases and using quick sum it is expected. But the result is strange (Figure 2)…

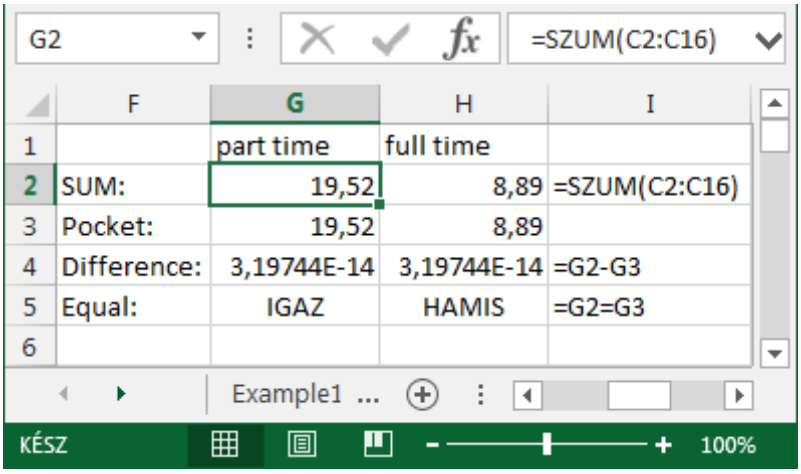

Figure 2. *Equality problem in Excel*

We can retype all numbers, functions and formulas… the difference is not zero. Checking the result, we get 19.52 is equal to 19.52 but 8.89 does not equal to 8.89… How can we use Excel if a very simple task like this is too difficult for it?

Students have to know, Excel is a program, and it implements all numbers binary, using limited bits. What does it mean in this case? What does it mean in general cases? Is it an MS specific problem? Solving the problem in Google docs (Figure 3) we can see the same.

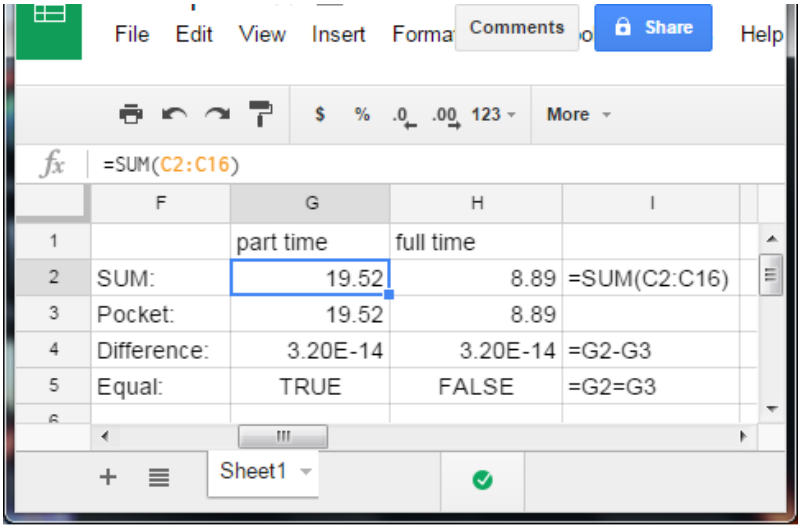

Figure 3. *Equality problem in Google spreadsheet*

We have to teach how computers work, how numbers are implemented in the program, what the similarities and differences between decimal and binary representation of numbers are and the effects in practice of finite representation. We have to talk about these questions while we teach how to use spreadsheets and we have to talk about these questions in connection with all applications which use float numbers (Figure 4.).

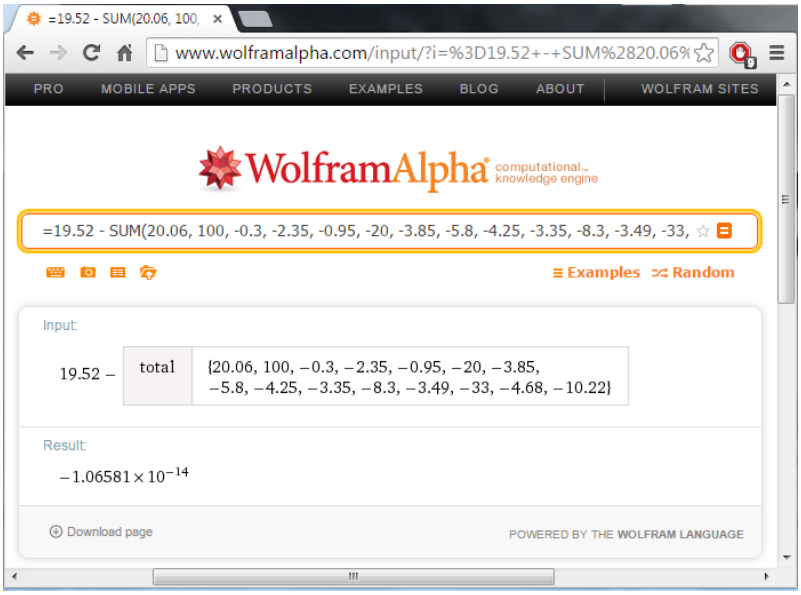

Figure 4. *19.52 –19.52 <> 0 by WolframAlpha*

Example 4 – Match(), VLookup(), HLookup() vs. Search

Advanced usage of spreadsheets is indicated by using match and lookups functions. We can find lots of videos and support to show how to use these function. These learning materials describe the rules learners should memorize. But memorizing is very difficult especially in long term. The most frequent problem is to keep in mind that the lookup array must be (ascending) ordered when you omit the match type or choose range lookup. The problem is more serious when the result is not verified but seems to be correct because to know something bad is worse than not to know.

In our approach the lessons focus on the Match() function and the main point is to clarify how this function works. Match function searches the lookup value in the lookup array, therefore we try to model how it is possible. Learners have to understand linear and binary search algorithms, they describe – and present – the effect of using binary search on unordered (or opposite sorted) list (Figure 5.), as well as they analyze the speed of search methods. Definitely these lessons are programming lessons connecting to a spreadsheet.

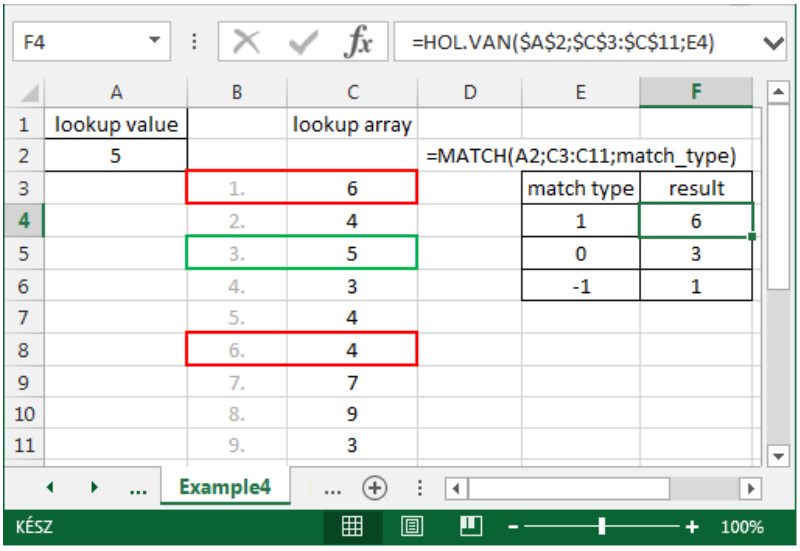

Figure 5. *Results of Match() function*

Understanding the algorithms hidden into a function, the usage is easier, the parameters' roles become understandable. Exploring binary search it becomes clear that the result is the row of a half-bounded interval's bound. Exploring linear search it becomes clear that Match() function with 0 as last parameter, looking from up to down or from left to right, finds the first exact match.

VLookup() and HLookup() are two special combinations of Match() and Index() functions. While we teach how to use these functions we have to elaborate why these functions are so useful why they are given separate names. What specialty has the first column or the first row in a range? We can give some impression of the role of unique identity keys and concepts of databases.

3. Benefit in Teaching Programming

Teaching applications as they are programs gives wide base to teaching programming. Every lesson of teaching spreadsheets provides an opportunity to teach programming, giving examples of data representation and usage of algorithms. Other applications like document and image editors help to understand other data structures and algorithms. Learning applications as program means that data are explored as binary representation, functions and methods are analyzed as algorithms. Learners have experienced concepts before they should use data and algorithms to produce a program.

4. Summary

Informatics education focuses on digital literacy and programming skills. In most cases teaching application and teaching programming are two disjunctive areas of lessons. We have presented a method to improve digital literacy of the students, meanwhile the teacher focuses on programming. Exploring spreadsheet (or other) applications students learn computational thinking, basics of programming. Teaching spreadsheets usually means teaching the usage of a software and teaching how to solve some typical problems with using spreadsheet. In our approach spreadsheet is a program with data and predefined algorithms. Students see a spreadsheet but they have to understand the program written by a programmer. We have demonstrated the method by four examples but it is a generalized idea to several other problems, methods and functions in spreadsheets, moreover to other applications. Teaching anything about application can lead to teaching programming indirectly.

Reference

[1] A. Cohen and B. Haberman (2010), "CHAMSA: five languages citizens of an increasingly technological world should acquire," *ACM Inroads*, vol. 1, no. 4, pp. 54--57.

[2] Y. Hofoku, S. Cho, T. Nishida and S. Kanemune (2013), "Why is programming difficult? - Proposal for learning programming in ”small steps” and a prototype tool for detecting ”gaps”," in *Informatics in Schools. Sustainable Informatics Education for Pupils of all Ages*, Potsdam, Springer.

[3] P. Hubwieser (2004), "Functional Modelling in Secondary Schools Using Spreadsheets," *Education and Information Technologies*, vol. 9, no. 2, pp. 175—183.

[4] P. Hubwieser, M. Berges, J. Magenheim, N. Schaper, K. Bröker, M. Margaritis, S. Schubert, L. Ohrndorf and Berdes (2013), "Pedagogical content knowledge for computer science in German teacher education curricula," in *ACM, WiPSE '13 Proceedings of the 8th Workshop in Primary and Secondary Computing Education*.

[5] Informatics Europe & ACM Europe Working Group on Informatics Education (2013), "Informatics education: Europe cannot afford to miss the boat,". [Online]. Available: http://europe.acm.org/iereport/ACMandIEreport.pdf. [Accessed 1. november 2014.].

[6] S.P. Jones, B. Mitchell and S. Humphreys (2013), "Computing at school in the UK". [Online]. Available: http://research.microsoft.com/en-us/um/people/simonpj/papers/cas/computingatschoolcacm.pdf.

[7] M. M. Sysłlo and A. B. Kwiatkowska (2013), "Informatics for all high school students: a computational thinking approach," Informatics in Schools. *Sustainable Informatics Education for Pupils of all Ages*, vol. 7780, pp. 43--56,

[8] M. M. Sysło (2014), Informatics for all students - A Computational Thinking Approach.

[9] P. Szlávi és L. Zsakó (2003), „Methods of teaching programming 1(2)", *Teaching mathematics and Computer Science*, %1. vol 01, pp. 247-258.

[10] J. M. Wing (2008), "Computational thinking and thinking about computing," *Philosoohical Transactions of The Royal Society A*, vol. 366, no. 1881, p. 3717–3725, 28 10.

[11] J. M. Wing (2006), Computational Thinking and CS@CMU, Carnegie Mellon University.

Resources

[12] Department for Education, GOV.UK (2014.), "National curriculum," [Online]. Available: https://www.gov.uk/government/collections/national-curriculum. [Accessed 1. November 2014.].

[13] Magyar Kormány, Nemzeti Alaptanterv (NAT), *Magyar Közlöny* vol 66, 2012.

[14] Exam Dividers, "ECDL - Excel 2010 - Diagnostic Question Assistance," *Exam Dividers*, http://edividers.co.uk/index.php/ecdl-excel2010-diag, 2015.

[15] IIT Bombay, "EDX - IITBombayX: CS101.1x Introduction to Computer Programming, Part 1," IIT Bombay, [Online]. Available: https://courses.edx.org/courses/IITBombayX/CS101.1x/2T2014/info. [Accessed 29. October 2014.].

[16] D. J. Malan, "EDX HarvardX: CS50x Introduction to Computer Science," Harvard, [Online]. Available: https://www.edx.org/course/introduction-computer-science-harvardx-cs50x. [Accessed 29. October 2014.].

[17] Microsoft (2006), *101 Ideas for Innovative Teachers*, Budapest: Jedlik Oktatási Stúdio.

[18] Z. Reményi, G. Siegler and Z. Szalayné Tahy (2011), *Informatika érettségire felkészítő feladatgyűjtemény*, Budapest: Nemzeti Tankönyvkiadó.

[19] Zs. Szalayné Tahy, *Feladatgyűjtemény a táblázatkezeléshez*, Budapest: Nemzeti Tankönyvkiadó, 1998.

[20] B. Zomick, "9 Excellent Resources to Help You Learn Excel Online," *Skilledup*, http://www.skilledup.com/articles/courses-learn-excel-online, 2013.

Authors

Zsuzsanna Szalayné Tahy, Eötvös Loránd University, Faculty of Informatics, Budapest, Hungary, e-mail: sztzs@caesar.elte.hu