

Programming for All?! A problem-based approach.

Introduction

If the motto „Programming for All!“ is taken seriously, it is not only children and adolescents who should become digitally competent. Especially adults whose school days did not include computers and the internet should acquire these competences, not only in order to understand their children's school material, but also in order to be able themselves to act competently even in a digitally influenced society.

These addressees are usually not very interested in the typical (mathematically and informatically influenced) introductory examples of school teaching. For these addressees, I present a curriculum that covers important basic concepts of informatics and programming - orientated towards working on motivating problems. Some examples will be presented in more detail: Coded Art, Animated Optical Illusions, and Logo Classics.

Three lead questions

Programming for everyone! At first, that sounds very general and today it is no more very original to start with the remark that digitization now affects all areas of life, our everyday life, our work and our social life. But it gets quite controversial and exciting when we ask ourselves how people are prepared for these changes in their world.

There are two main questions - at least as far as I follow the discussion in Germany:

1. The first question is: What should actually be taught? Are these the basics of computer science or is it media literacy? And should they be taught in a separate subject computer science or integrated with all subjects?
2. Just as controversial is the second question: When should it all start. A few are in favor of the kindergarten, some for elementary schools and most for secondary schools. A few even want to keep the school completely free of them.
3. But for me, a 3rd question also arises. How do we reach all those who have already left behind them their school and training phase, but who lack such basic education (in fact, this is by far the largest of all target groups)?

A curriculum for adults

For some time now, I have been engaged in working with older people and helping them with their problems with digitization. A part of this group also would like to understand their digital environment. That's why I started a search, focussing on computer science, for suitable concepts and materials that could cover the corresponding basics.

Because computer science lessons should not be oriented towards short-term developments, there are several attempts to define the central concepts of computer science. Zandler & Spannagel (2006) for example tried to determine them empirically by interviewing experts.

One of their results has been, that the importance of the content concepts are judged quite differently, but with an outstanding position of the concept algorithm. In the process concepts on the other hand, analyzing and problem posing and problem solving clearly dominate.

Overall, this is a strongly academic viewpoint. So in addition, I also looked for concrete learning materials. Unfortunately, the usual offers rarely help.

- Books as Introductions to computer science or programming are usually aimed at students (e.g. Brookshear & Brylow, 2014).
- Some books address specifically adults and the elderly. But they are mostly limited to pure applications(e.g. Ewin, 2017).
- Books for children, on the other hand, usually focus on games, which is not necessarily interesting and suitable for older people (e.g. Briggs, 2012).
- Then there is adult education, especially by the Volkshochschulen (I'm not sure if in other countries the so-called „folk high schools“ are the correct counterpart to it). At least the Volkshochschulen in my surrounding area only offer application trainings.
- Then, of course, online courses are still to be considered. The offer is extremely varied and confusing, but also here offers for students dominate (an interesting exception is the course „Informatik für Einsteiger (2019 edition)“, Modrow, 2018).

Above all, I have missed the orientation towards personal interests of the target group. How necessary this is was already formulated by Brian Harvey years ago in the foreword to his very special Logo books (Harvey, 1997, p. xii):

„See, the wonderful thing about computer programming is that it is fun, perhaps not for everyone, but for very many people. The bad news is that the curricula tend to be imitations of what is taught to beginning undergraduate computer science majors, and I think that's too rigid a starting point for independent learners. The ideas of computer science are a means to the end of getting computers to do what you want.“

This statement is in accordance with my experience that it is more important for our addressees to apply the computer science content in the areas they are actually interested in. It is no surprise then that Brian is partly responsible for the concept of the computer science course bjc - the Beauty and Joy of Computing course - to which I owe many ideas for my own concept.

Big Ideas: Things to learn	Computational Thinking Practices: Things to do
Creativity	
Abstraction	Connecting Computing
Data and Information	Creating Computational Artifacts
Algorithms	Abstracting
Programming	Analyzing Problems and Artifacts
The Internet	Communicating
Global Impact	Collaborating

Table 1: Seven „Big Ideas“ and six „Computational Thinking Practices“

The curriculum framework of the course is organized around seven "Big Ideas" and six "Computational Thinking Practices" (AP Computer Science Principles, 2017, pp. 9-34; see

also Grabowski & Nake, 2019). Naturally there are strong overlaps. Abstraction is the idea, Abstracting is the practice etc. According to the authors Creating is by far the most important of the Practices, and the most important artifacts are computer programs!

For me it was very helpful that the authors also mention the big ideas of programming. Some are listed here:

- Variables and Scope
- Iteration
- Lists
- Parallelism
- Event Handling
- Procedures (Commands and Reporters)
- Recursion
- Functions and Lists as First Class Data

Learning to program then means applying these concepts properly in the respective context. As an interim goal, it remains to be noted that it will be a question of connecting personal interests with learning and applying those "Big Ideas". Therefore, in terms of Papert's constructionism, it is the use of the computer or digital tools in general to support design processes.

A dedicated constructionistic-psychological approach „learning by design“ was presented by Lehrer, Erickson & Connell (1994). They name four central types of cognitive activity in design, namely planning of a product, implementing the design idea, rating and revising the product. My approach is based on these three points:

1. The meaningful artifacts will be computer graphics.
2. Any new idea is introduced and implemented with a specific graphic.
3. The design process as such consists of the recoding & remixing of meaningful examples.

Three examples

On this basis, I would like to concretise the concept using three examples and show what results can be achieved with it. Let's start with the first point, the „meaningful artifacts“. Today the influence of digital media on the production and representation of graphic elements is enormous. It is therefore obvious to choose graphics as the topic of introduction to programming.

I personally love graphics, especially abstract geometric graphics. That's why for me the creation of graphics is a very central and motivating application of the computer. Currently I am working on three different such topics:

The first is Logo Classics: Turtle graphics is an essential part of Logo since its beginnings. In early publications (like handbooks or introductory books) you can always find very similar graphics that have their own aesthetics. I am using step by step the „big ideas“ to build up a collection of those graphics. Unlike the original ones, all examples here are animated and interactive.

The same applies to my second topic, Computer Art or more general Digital Art. Computer Art is any art in which computers play a role in the production or the display of the artwork. Here, too, we go beyond static models by animating them and control them interactively.

The third field of application are optical illusions. Again, the unique feature are the interactively controlled animations.

The second point in my concept is absolutely central: Each idea is presented with a graphic example, which only requires the introduction of this one new idea.

As in all projects graphics will be produced, a prerequisite for all graphics is then naturally a set of turtle graphics commands, like *move* or *turn*. Fortunately these are very intuitive with its natural geometry, which Seymour Papert called "body syntonic" (Papert, 1980, p. 63) - and that works quite well even with adults! So that's the starting point for any implementation.

Animated and interactive projects

A first simple example is the Oppel-Kundt-illusion: A line with transverse strokes seem to appear longer than a line of the same length without such transverse strokes. For this graphic, in addition to the graphics commands, it really only needs repeat loops.



Figure 1: Oppel-Kundt-illusion (right) and code snippet with loop and variables (left)

Characteristic of algorithmic images in general is the precise, rule-guided arrangement of basic elements. The single picture is therefore always a copy of a "class" of many more comparable pictures (Grabowski & Nake, 2019, p. 85). The programs should therefore be as flexible and interactive controllable as possible. To this end, variables can be introduced (as shown in the example).

If this approach is continued consistently, this results in a coherent overall concept almost by itself. Table 2 shows an incomplete compilation of the „new ideas“ and the related graphic examples.








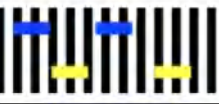

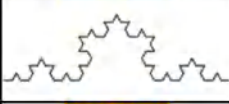


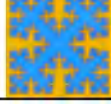
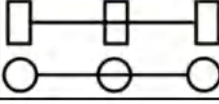

	Logo Classics	Opticals	Digital Art
iteration			
procedures & variables			
lists (of lists)			
recursion			
procedures as data			

Table 2: Examples for „One idea – One Product“

It is possible with the concept of iteration alone to produce the circular lines (Logo Classics), the Opper-Kundt- illusion (Opticals) or the picture Rhythms as an homage to Erwin Steller (Digital Art). In the same way, the use of other concepts will be successively introduced using concrete images. In the case of the Logo Classics you need procedures and variables for the nested polygons or lists in the case of repeated open polygons (Harvey, 1997, pp. 186).

The third point in my concept is learning by design or rather recoding and remixing. You remember? It is the planning, implementing, rating and revising a design idea. The general rule is: Think first, code second!

For me, the following procedure has proven to be successful:

- first determine the required graphical elements
- which ones are to be animated
- and finally, which parameters of the graphics should the viewers be able to change

In the first step, we can follow the swiss artist Ursus Wehrli (2003), who started the project tidying up art. He has inspired many teachers to open up access to works of art and art movements in this way in art lessons.



Figure 2: Modified Café Wall illusion (left) and required image elements (right)

In the same way we can analyze our image templates and determine the required image elements. The example in Figure 2 shows a modified Café Wall illusion by Akiyoshi Kitaoka. He is a psychologist, who is world-famous for his novel optical illusions. We see that we only need black and blue lines and a lot of chessboard patterns in black and white. It should be possible that the viewer can choose the thickness and the color of the lines themselves (this is the interactive component) and that they can freely rotate the chessboard patterns (this is the animated component).

The animation principle corresponds to traditional Stop-motion technique. Each image is continuously replaced by a new image with updated characteristics (such as colors, lengths, angles, etc.). So it's the principle paint – wipe – paint ... For the viewers to be able to interactively control the illusions, sliders are provided for the characteristic values. The image then will be updated with them at the next repeat loop.

Conclusion and Outlook

When selecting suitable projects for the implementation of my approach, it helps to orient them to existing examples. Citation, copying and alienation have long been part of an artistic practice, the so-called "Appropriation Art". On the one hand, this orientation helps to obtain appealing and motivating images from the very beginning. On the other hand, they provide suggestions to approach your own variants and developments.

The choice of Snap! as a development environment was not accidental under these circumstances. Amongst other things it offers any stage sizes, assembling nested sprites, or sending messages to individual sprites. For me personally it was important that I thus could not only work playfully with it, but that serious products could be created. I created pictures for exhibitions and multimedia festivals. Some versions of the optical illusions can serve as a laboratory with which serious experiments can be replicated.

The application of my concept on other topics is quite possible. I am thinking of the examination of architectural styles (see e.g. Schweiger, 1992). It is also obvious to model simple dynamic systems or agent-based systems.

References

- Brigs, J.R. (2012). Python for Kids: A Playful Introduction to Programming. San Francisco: No Starch Press
- Brookshear, G. & Brylow, D. (2014). Computer Science: An Overview. Boston: Pearson
- Ewin, C. (2017). Computers for Seniors. San Francisco: No Starch Press
- Harvey, B. (1997). Computer Science Logo Style. Vol. 1: Symbolic Computing. London, England: The MIT Press
- Grabowski, S. & Nake, F. (2019). Algorithmische Kunst als Bildungsgegenstand. Gedanken zu einer fachlichen Bildung über Fächer hinaus. MedienPädagogik: Zeitschrift für Theorie Und Praxis Der Medienbildung, 33 (Medienpädagogik und Didaktik der Informatik), 76-101.
- Lehrer, R., Erickson, J. & Connell, T. (1994): Learning by Designing Hypermedia Documents. In: Computers in Schools, Vol.10 (1994), I. 1/2, 227 – 254
- Modrow, E. (2018). Computer Science with Snap!. Scheden: emu-online. <http://www.emu-online.de/ComputerScienceWithSnap.pdf>
- Papert, S. (1980). Mindstorms. : children, computers, and powerful ideas. New York: Basic Books
- Schweiger, P. (1992). Gotik in Pascal. München: Oldenbourg
- Wedekind, J. (2018). Codierte Kunst – Kunst Programmieren mit Snap! Tübingen: self-published
- Wedekind, J. (2019). Optische Täuschungen animieren für Dummies Junior. Eine Einführung mit Snap! Weinheim: Wiley CH-Verlag
- Wedekind, J. (in press). Logo Classics! Tübingen: self-published
- Wehrli, U. (2003). Tidying Up Art. New York: Prestel
- Zendler, A. & Spannagel, C. (2006). Zentrale Konzepte im Informatikunterricht: eine empirische Grundlegung. Notes on Educational Informatics — Section A: Concepts and Techniques 2 (1): 1–21