ICT-Rich Programming Projects

Michael Weigend

ICT-Rich Programming Projects

Michael Weigend

Holzkamp-Gesamtschule, Willy Brandt Strasse 2, 58452 Witten, Germany
mw@creative-informatics.de

Abstract. This contribution advocates designing programming lessons in a way that students use information and communication technology (ICT) extensively. The paper presents four examples of such ICT-rich programming projects with different levels of required programming expertise: 1) Write directions for walking from one place to another using Google Maps and Streetview; 2) Develop a Python program that creates a text using words and phrases from free literature; 3) Write a program that creates a list of words representing controversial issues from automatically generated interview transcripts; and 4) Create a program that analyses a comma-separated values (csv) file with the results of a self-made Google Forms survey. The examples illustrate benefits of the combination of ICT and programming: students discover new ICT functionality and get a deeper understanding of digital technology. They experience that programming knowledge empowers uses of digital technology in new ways. Having the opportunity to use ICT tools may motivate teenagers to go deeper into computer science.

Keywords: ICT, Programming, Computer Science Education.

1 ICT and Computer Science Education

There is a permanent discussion about what to teach in Computer Science (CS) classes in schools. According to Webb et al. [1], there is a consensus in the community of educators and curriculum developers that digital literacy – the skills to use information and communication technology (ICT) – and CS are complementary and that both are needed in the school curriculum. CS curricula usually focus on principles and practices related to *creating* digital artefacts. In the CSTA K-12 CS standards [2], basic competences related to *using* ICT are mentioned just for level 1a (ages 5–7 years), like “Select and operate appropriate software to perform a variety of tasks” (1A-CS-01) or “Store, copy, search, retrieve, modify, and delete information using a computing device“(1A-DA-05). At higher levels, the focus is on creating, understanding and evaluating digital technology. Among the 102 competences listed for levels 1b to 3b there are only 3 on using digital tools.

Bridging ICT and CS can be done in different ways. Brinda et al. [3] suggest connecting basic CS concepts like finite state automata and object-oriented modelling (OOM) with ICT usage. State transition diagrams may help understanding of how to use audio players, and OOM concepts help understanding of the philosophy of text editors. For example, when Tina sees characters and paragraphs as objects of different types, it might be easier for her to understand how to change attributes (like size and colour of characters and line spacing and alignment of paragraphs). Dagienė [4] suggests that activities like “Computer Science Unplugged” [5] and the tasks of the Bebras Challenge on Informatics and Computational Thinking lead to a deep understanding of ICT. Although the Bebras tasks cover ICT as a topic, they require little or no ICT usage. All information necessary to solve the task can be found in the task description and the children can principally solve the tasks in their minds or with pencil and paper.

In contrast to this approach, this paper focuses on ways to encourage ICT usage for creative programming projects. Obviously, students use an integrated development environment (IDE) in each programming project. But beyond that, in CS lessons, often only a little technology use is required. There is a huge supply of free resources on the Internet that can be used for programming projects: images, movies, data, texts and digital tools (apps). In an ICT-rich programming project, these resources are used extensively for creating and collecting data that are needed for programming projects.

2 Example Projects

In this section, I present four projects with extensive usage of ICT, sorted according to required programming skills.

2.1 Visiting Cool Places

An algorithm is the precise description of instructions to solve a problem. An example is the directions guiding a person from one place in a city to another. If the directions are correct and correctly executed, the person following the direction arrives at a certain place. The first ICT-rich project is to create directions using Google Maps. This task does not require any specific pre-knowledge and has been used as part of an introduction into algorithms with 14-year-old students. It is probably also suitable for younger age groups.

Task 1: Pick a place on Google Maps somewhere in the world. Write a formatted text document, with images, that explains how to walk from this place to another place by using Google Street View. Street names are only allowed for describing the starting point.

At the end of the text, ask the reader a question that the reader can only answer, if she or he has arrived at the correct place.

First, try this mini example.

Open Google Maps and go the Brandenburger Tor in Berlin. Put the orange Street View Pegman on the Pariser Platz. Turn around until you see the Brandenburger Tor right in front of you. Walk through the Brandenburger Tor on the right-hand Street View path. Immediately behind the Brandenburger Tor, there is something on the right-hand side. Which of these items can you see on your right-hand side immediately behind the Brandenburger Tor?

A: Wastepaper basket; B: Manhole; C: Spotlight; D: Traffic sign.

To create their own directions, the students are encouraged to use Google Maps, Google Street View, text editing and image editing tools. Some students tend to use directions that are then automatically generated by Google Maps. To avoid this, the task contains the rule that position names of streets may be used only for the specification of the start. The general idea is to write down directions you would give to a person that has asked you for the way to a certain place.

Regarding the algorithmic nature of this task, the students face three challenges:

- The starting point and the direction in which the reader has to look first must be explained very precisely using information that is available on Google Maps.
- On the way, they have to exploit Street View images to find appropriate landmarks that are easy to describe and easy to find. A challenge is to find good wording. Frequently used phrases are: “Turn around until you see ....”, “Walk along the road until …” indicating repetitions.
- They have to create questions for checking whether or not the reader executes the algorithm correctly. At the same time, the reader’s correct answers indicate that the algorithm (directions) is correct. Checking correctness is a computational competence. Computer programs may contain assertions that guarantee correctness to some extent. In this case, the students need to use ICT to find good questions. Ninth-graders from a German high school, who did this project, had these ideas: Tina (name changed) created a section from a Google Street View screenshot at the goal, depicting a statue, removed the text using Windows Paint and asked: “What is written on the statue?” Tom asked the reader to use the Google Maps tool for measuring distances and to measure the length of a house at the goal.

2.2 Mining Mark Twain

The second project is an example of creative coding exploiting textual data that are available on the Internet. In Project Gutenberg (http://www.gutenberg.org), one can find more than 60,000 free e-books, including the complete work of Mark Twain (5,598 pages). Creative coding (CC) is developing computer programs for personal expression or artistic purposes. CC has been used in CS education for many years [6, 7, 8]. Pioneers in CC are artists like Hiroshi Kawano (China, Japan) and Francois Morellet (France), who have used computers to generate paintings since the early 1960s. In 1960-61, Morellet used textual data to create a painting. He distributed 40,000 squares according to even and odd numbers in a telephone directory (Zentrum für Digitale Kunst in Karlsruhe, Germany). In literature, there are sonnets, haiku, or other forms written by computer programs [9]. The website “Bot or Not” (http://botpoet.com/) offers a Turing Test for poems. Chris Peck and Eleanor Bauer developed the text of the theatre play “New Joy” (premiere 2019 at the Schauspielhaus Bochum, Germany) using simple algorithms. The following task has been used in introductory programming classes (secondary school and university) in the context of string processing. To be able to understand and to improve the starter program, the students should have basic programming skills (control structures, files, strings and lists).

Task 2: Write a Python program that creates a short text (say for a birthday card) picking parts from the complete work of Mark Twain.

The following starter program illustrates basic Python techniques, that can be adopted. The text file containing Mark Twain's work must be downloaded and stored in the project folder. The program reads this file and creates a string object named `text`. The statement in line #2 creates a list of sentences (strings ending with ".") In the while-loop, items from this list of sentences are randomly picked again and again until a sentence with a length between 20 and 50 characters is found. Three of these randomly chosen sentences are concatenated to a string.

```
from random import choice
f = open("marktwain.txt", encoding="utf8")
text = f.read()                                         #1
f.close()
sentences = text.split(". ")                            #2
result = ""
for i in range(3):
    sentence = " "
    while not(20 < len(sentence) < 50):                 #3
        sentence = choice(sentences)
    result += sentence + ". "
print(result)
```

Output (example):

```
He was competent and satisfactory. The matter ended in a
compromise, I submitted. Still he could not understand.
```

The basic idea of a starter program is that students copy it, run it and change it until they understand it completely. Then they improve and extend it, for example:

- Add documentation/trancing features: how many random picks were necessary? How many sentences of a certain length contains Twain's work?
- Develop/refine the functionality: search for sentences that contain certain words (e.g. the first name of a friend); write computer poems, using words that rhyme.

More examples for simple Python projects exploiting textual and visual data from the Internet (twitter tweets, webcam life images, etc.) can be found in Weigend [10].

2.3 Evaluating Interviews

What are hot topics and controversial issues in the school community? In a classroom project at a German high school, students tried to find out, and interviewed school friends and teachers using their smartphones. The goal of the interviews was to get a collection of words indicating controversial issues of school life.

The students developed a small set of questions that were supposed to evoke statements on conflicts (e.g., do you remember a conflict in the classroom today? Did something on the schoolyard bother you today?) They interviewed each other first and then interviewed teachers and students on the school campus using their smartphones and a voice recorder app with voice-to-text functionality. The text files (created from audio) were collected in a folder. This can be done by sending the files to a common Google Drive Folder, which has been created and made accessible to everybody by one member of the group. Each student copies these text files with the interviews into a subfolder named "data" in his or her project folder.

The project is suitable for Python programming classes in secondary education, especially if the school supports interdisciplinary teaching. The students should be familiar with sequences and sets. As part of a computer science curriculum, the project can serve as an example for modelling with dictionaries.

The starter program illustrates basic Python programming techniques with files, dictionaries, and sets. The program reads all files in the data folder and creates a string containing all interviews (the loop starting in line #1). It splits the string into a list of words and creates a set of words by eliminating duplicates (#2). Using this set and the original list of words the program creates a dictionary, which maps each word from the set to its frequency (the loop starting in line 3).

```
import os
interviews = ""
for fn in os.listdir("data"):                           #1
    f = open("data/" + fn, encoding="utf-8")
    interviews += f.read()
    f.close()
words = set(interviews.split())                         #2
d = {}
for word in words:                                      #3
    d[word] = interviews.count(word)
print(d)
```

Example output (shortened):

```
{'jetzt': 6, 'brauchen': 1, 'sein': 12, 'schließen': 1,
'a': 468, 'Überzeugung': 1, …}
```

There are several approaches for extension and improvement, for example:

- Improve the usability: the program should filter the set of words and ignore irrelevant words like numbers or pronominals. A simple way to filter is considering only words with more than five letters. Students who are familiar with the Python interface Tkinter may create an application with a graphical user interface that displays the most relevant issues in a comprehensive way (see Fig. 1).

- Improve the technical quality: define functions or classes and split the program into smaller parts.

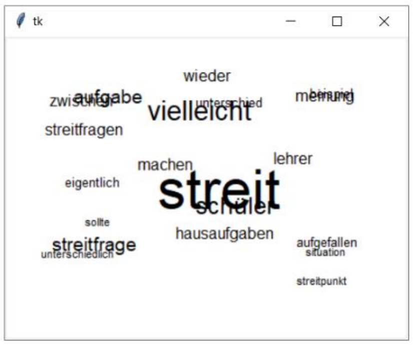

Fig. 1. Words with a minimal length of five letters are displayed on a canvas at random locations. The size of the words depends on the number of occurrences in the interviews.

2.4 Evaluating a Google Forms Survey

With the free version of Google Forms, students can conduct surveys. They create a questionnaire, make it accessible in the Cloud and use it for interviews on the schoolyard. With the free version of Google Forms it is possible to get an overview of the answers. Although the questionnaire might contain some personal questions like age and gender, unfortunately, filtering is not supported. For example, it is not possible to compare the responses of girls and boys. However, it is possible to create a comma-separated values (csv) text file. The csv text file is structured like this:

- The first (very long) line contains the questions of the questionnaire as strings separated by commas. There is no comma at the end of the line.
- The other lines contain the answers given by the people who completed the questionnaire. If a respondent has not answered a question, this non-answer is represented by an empty string.

Consider this mini survey:
 Question 1: Which of these statements do you agree with?

- Listening to music should be allowed (wearing headphones), while quietly working in the classroom.
- The first lesson should not start before 9:00.
- There should be a vending machine for snacks in the lobby.

Question 2: What is your gender?

- Female
- Male

- Diverse

The csv file starts with two lines similar to these:

```
"Time Stamp", "Which of these statements do you agree
with?", "What is your gender?"
"2019/09/17 8:54:20 AM OEZ", "The first lesson should not
start before 9:00.; There should be a vending machine for
snacks in the lobby.", "Male"
```

Each line ends with the invisible new-line character (escape sequence: `\n`). Since lines in a csv file can be very long, on paper it might require several lines to print a single line from the file. In the example, the second data line starts with an (automatically generated) time stamp, indicating when this response was submitted. The second string contains the two checked options of the first multiple choice question separated by a semicolon. The third string of this line is the selected gender.

Unique Options Method. The example questionnaire was designed in way that makes it very easy to program an analysing tool. Each option is unique. It occurs only once in the questionnaire. That implies that the data analysis can be done simply by counting occurrences of strings.

This programming project should be a part of an interdisciplinary project in upper secondary education. To be motivated for the program development, students must be involved in the design of the underlying research questions.

The following listing is a starter program that students can test, modify and extend (a few lines are shortened for printing reasons). It illustrates a few basic techniques:

- Reading the csv file and creating a list of lines containing the answers of all participants (#2).
- Filtering the data by generating separate lists with data from female and male participants adopting Python list comprehensions (#3).
- Generating a table with columns for output.
- Counting the occurrences of options (using substrings to keep the program shorter) in the two data lists of female and male respondents (#4).

```
LINE = '{:45} {:6} {:6} {:6} '
STATEMENTS = ['It should be allowed to listen to music',
      'The first lesson should not start before 9:00',
      'There should be a vending machine for snacks']  #1
f = open('test_csv.txt', encoding='utf8')
data_lines = f.readlines()                             #2
f.close()
male = [resp for resp in data_lines if 'Male' in resp]
female = [resp for resp in data_lines
                if 'Female' in resp]                   #3
print('Statement                   Female  Male   All')
print('-----------------------------------------------')
```

```
for stat in STATEMENTS:
    output = LINE.format(stat,
                         str(female).count(stat),
                         str(male).count(stat),
                         str(data_lines).count(stat))     #4
    print(output)
print('n = {}'.format(len(male) + len(female)))
```

Example output:

```
Statement                                       Female  Male   All
-------------------------------------------------------------------
It should be allowed to listen to music            10     11     21
The first lesson should not start before 9:00       2     10     12
There should be a vending machine for snacks        6     12     18
n = 21
```

The starter program can be changed and extended by the students in a number of ways:

- Adapt the starter project to a different – self-made – survey.
- Make it interactive. The user could choose the filter criterion.
- Add statistical features (tests).

Parsing the CSV Text. The example program from the previous section only works with very special questionnaires. Programs that are supposed to handle every kind of survey need to parse the csv text. This requires more programming experience. One approach would be to create a list of dictionaries, each dictionary representing the data of one participant in this format:

```
{question1:answer1, question2:answer2, …}
```

The key of each item (question1, …) is a string containing a question from the first line of the csv text. The value of each item (answer1, …) is a tuple of strings, representing the answers, given to the question.

3 Conclusion

This contribution presented four examples of ICT-rich programming projects. In these projects, the students experience that programming competence implies empowerment. With programming knowledge, it is possible to use digital tools (e.g., Google Forms) and other resources like free literature in new ways.

The second benefit from ICT-rich projects is that students discover new digital resources and new features of tools that they already use every day. For example, when students write precise directions based on Google Maps and Street View, they use different functionality from that when looking up a bus connection.

This advanced usage of digital tools also implies reflecting their potential and limitations. In the case of the Google Maps project, students reflect on the difference between directions using landmarks that are easy to find and to describe, and automatically generated directions.

Finally, students get a deeper understanding of the internal structure of digital resources. For example, when programming a tool for analysing Google Forms survey data, they become aware of the structure of csv representations.

ICT-rich programming projects can easily be integrated into a conventional CS curriculum that covers coding-oriented competences like those in the CSTA standards [2]. The usage of ICT is not systematically introduced, but teachers assume that students already know most of the tools and can learn things they do not know yet *en passant*. This fits with the idea that the CS curriculum focusses on principles and concepts that are long-lasting and difficult to learn, whereas the elements of digital literacy change quickly and are easy to learn. Many apps that people download and use do not require a systematic introduction. ICT-rich programming projects encourage the students to improve their digital literacy on their own. The ideal environment for this is a Parkhurstian laboratory, fully equipped with relevant literature and a teacher that can give expert support through a self-controlled learning process.

In students' perceptions, there is a strong connection between ICT and CS. Students who code are also interested in ICT. The results of an Estonian study suggest that programming knowledge from high school education motivates students to enrol in ICT subjects (not just computer science) at university [11].

Fluck et al. [12] have stated economic, social and cultural rationales for computer science as a school subject. These rationales are not only relevant in the general discussion of school development; they are also relevant for teenagers seeking their way into society by developing views, and planning their professional future. ICT-rich programming projects illustrate the relevance of computer science in the economic, social life culture in a modern society, and might influence students' career decisions. Here it is not just *claimed* by the teacher that CS is relevant, but students actually experience the benefits of CS concepts while working with ICT.

Many students love science not just because of the concepts they learn, but because they have the opportunity to get exciting experiments to work with real chemicals and laboratory equipment. In the same way, the combination of using interesting ICT and programming (including learning fundamental CS concepts) might motivate teenagers to engage in CS.

References

1. Webb, M., Davis, N., Bell, T., Katz, Y.J., Reynolds, N., Chambers, D.P., Sysło, M.: Computer science in K-12 school curricula of the 2lst century: Why, what and when? Education and Information Technologies 22(2), 445-468 (2017).
2. Computer Science Teachers Association.: CSTA K-12 Computer Science Standards, Revised 2017. https://www.csteachers.org/page/standards, last accessed 2019/10/29.

3. Brinda, T., Puhlmann, H., Schulte, C.: Bridging ICT and CS: educational standards for computer science in lower secondary education. ACM SIGCSE Bulletin 41(3), 288-292 (2009).
4. Dagienė, V.: Challenge to Promote Deep Understanding in ICT. Paper presented at the 6th IFIP World Information Technology Forum (WITFOR), San José, Costa Rica, pp.47-52 (2016).
5. Bell, T., Witten, I., Fellows, M.: Computer science unplugged. Department of Computer Science, University of Canterbury, Christchurch, New Zealand (2002).
6. Greenberg, I., Kumar, D., Xu, D.: Creative coding and visual portfolios for CS1. In: Proceedings of the 43rd ACM Technical Symposium on Computer Science Education, pp.247-252 (2012).
7. Peppler, K., Kafai, Y.: Creative coding: Programming for personal expression. In: Proceeddings of the 8th International Conference on Computer Supported Collaborative Learning (CSCL), Rhodes, Greece, Volume 2, pp.76-78 (2009).
8. Wood, Z.J., Muhl, P., Hicks, K.: Computational Art: Introducing High School Students to Computing via Art. In: Proceedings of the 47th ACM Technical Symposium on Computing Science Education, pp.261-266 (2016).
9. Funkhouser, C.: Digital Poetry: A Look at Generative, Visual, and Interconnected Possibilities in its First Four Decades. In: Schreibman, S., Siemens, R. (eds.) A Companion to Digital Literary Studies. Blackwell, Oxford (2008).
10. Weigend, M.: Making Computer Science Education Relevant. In: Proceedings of the Information and Communication Technology-EurAsia Conference, pp. 53-63. Springer, Cham, Switzerland (2015).
11. Kori, K., Pedaste, M., Leijen, Ä., Tõnisson, E.: The role of programming experience in ICT students' learning motivation and academic achievement. International Journal of Information and Education Technology, 6(5), 331-337 (2016).
12. Fluck, A., Webb, M., Cox, M., Angeli, C., Malyn-Smith, J., Voogt, J., Zagami, J.: Arguing for Computer Science in the School Curriculum. Educational Technology & Society, 19 (3), 38–46 (2016).