Starter Projects in Python Programming Classes

Michael Weigend

Starter Projects in Python Programming Classes

Michael Weigend

Westfälische Wilhelms-Universität, Corrensstr. 80, 48149 Münster, Germany
mw@creative-informatics.de

Abstract. Starter projects are part of constructionist learning arrangements (1, 2). The term was mainly coined by the Scratch community and describes a simple, executable program that is designed to encourage copying, exploring and further development. This contribution first discusses three types: test programs, examples that illustrate programming techniques and algorithms, and architectural prototypes. It then presents self-observations of university students, describing how they use starter projects for learning.

Keywords: Starter project, programming, constructionism, Python

1 Introduction

1.1 Starter Projects

A starter project is an executable computer program that has been designed as learning material. Beginners copy, test and change the starter project, and develop it further according to their interests. The term starter project was coined and cultivated by the scratch community. There are collections of starter projects on various topics on the Scratch website. A well-known example is a minimal version of the video game Pong. The concept of the starter projects is based on the fundamental idea of constructionism ([1], [2]): People learn best when they construct relevant (digital) artifacts and share them with others. The term “starter” expresses that a development process is being started. This can be seen both as an individual and as a collective process. From the individual perspective, the starter project is the starting point for a series of iterations in which the software is refined, expanded and improved. This roughly corresponds to the procedure for agile programming (e.g. Extreme Programming [3]). A collective development process happens, when remixing Scratch projects takes place: Somebody picks a Scratch project that she or he likes from the Scratch website and develops it further in the online editor. In this way, a tree of derived projects (remix tree) is created from a starter project. For example, there were more than 26,000 remixes of the just mentioned starter project Pong (February 2021).

1.2 Starter Projects and Tinkering

A constructionist concept connected to starter projects is tinkering. “The tinkering approach is characterised by a playful, experimental, iterative style of engagement, in which makers are continually reassessing their goals, exploring new paths, and

imagining new possibilities." [4, p.164]. Resnick and Rosenbaum see spontaneous trying out and tinkering in contrast to instructional guidance in tutorials and detailed task descriptions that hinder an independent and self-controlled approach. They [5] point out that trying out also includes making mistakes and requires a "Failure-friendly way of thinking". However, failure is only helpful for learning if there is an opportunity to improve. The time frame of lessons is strict and children from socially disadvantaged backgrounds often do not have the opportunity to finish their projects at home [5]. A starter project makes it easier to start a tinkering process, and guarantees success because it already works. From the starter project, students can carefully find their (individual) ways into new areas, and try out unfamiliar programming concepts. The number and scope of the iterations can be determined by themselves. Which working environment do programming exercises with starter projects require? They [6] describe the general conditions for (physical) tinkering activities. Among other things, a seed, a playground (physical and organisational environment), tools and materials are needed.
Seed. The seed is the reason for starting the tinkering. This can be anything which arouses interest and motivates students to start. By definition, a starter project can be the seed of a software project.

Playground. The ambience of the room should support a playful attitude, cooperation and sharing results [7]. Terms such as Makerspace, Incubation Space or Fab Lab have been established for the architectural concept. Regarding the organisation, it is important that there is enough time for creating presentable products at every pace of work, and that there is opportunity to share results.

Tools. In programming projects, the primary tool is the development environment. In the case of Python programming, these are e.g. IDLE (Integrated Development and Learning Environment), which is included in the standard distribution, or other educational development environments such as Thonny and Geany.

Material. In a physical maker project, the materials that can be used are typically spread on a table. They invite participants to touch and try out. At the same time, they also represent a limitation or channeling, because the project can only be completed with the material offered, and with nothing else. In a programming exercise, the materials to create a digital artifact are the programming concepts used. A starter project can be a material in several ways. It explains and illustrates certain programming techniques, and it is also a model for good programming style (structuring, meaningful names, comments etc.). Sometimes, a collection of starter projects, each one illustrating a different technique, may be of help. Further materials that are brought in are language references, sheets with overviews ("cheat sheets"), skill cards, or posters on the walls.

2 Starter Projects and Examples

2.1 Types of Starter Projects

Principally, every computer program can be the starting point of a development, and can therefore be a starter project. However, in this paper it is assumed that a Starter Project has been designed specifically to encourage and facilitate learning. Regarding the intended learning activities one can roughly distinguish three types of starter projects: Test program, example, and architectural prototype.

Test Program. A test program is a small program with the primary function of demonstrating that the mechanics of program execution work. The classic example of a test program is a "Hello World program" that displays a short text like "Hello world" on the screen, and does nothing else. Probably the first Hello World program was published by Brian Kernighan in 1974 in a tutorial for the C programming language. The much-quoted original text reads

```
main() {
        printf("hello,world");
}
```

Although the program text is very short, it contains some abstract concepts and is not easy to comprehend. In Python syntax the code is much simpler:

```
print("Hello world")
```

However, if this is the very first program that a person writes and runs on the computer, it is not expected that they understand the program constructs. When a student just writes the program in the editor, starts it, and looks at the output, he or she has not understood the print() function better. But this person, who has never done this before, gets confident that he or she can do it. Exploration and trying out is not targeting the program text itself but its digital environment: The integrated development environment (IDE) (write a program text, understand the syntax highlighting) the operating system (create a project folder, store, load and run a program) and the interpreter or compiler of the programming language (understand system messages).

Test programs may be more complex than a "Hello World" program. Figure 1 shows a screenshot from the run of a simple Python program showing the live image from a public webcam (webcam viewer). When you click the button, the image is updated. You can see the changes. The source code is truly short (17 lines of code). However, to understand it requires specific programming knowledge (e.g., about GUI programming and image processing). The program is not suitable for introducing new programming concepts, such as the definition of button widgets. That would be done with different (more specific) examples. The program is a test program if the purpose of running it is to get certainty that it works in the current environment.

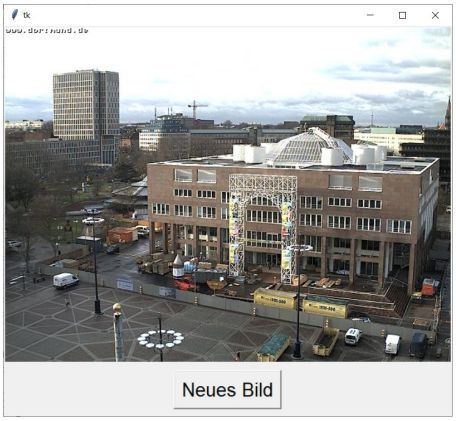

Fig.1. Webcam-Viewer

A test program like the webcam viewer can be used to initiate a creative task. Example: “Describe the idea for a Python project that evaluates live images from web cams and extracts meaningful information”. In case of the web cam image shown in Figure 1, a possible project idea is: The program checks whether someone tries to climb over the site fence. (The program checks changes in colour from the pixels on the site fence.)

The advantage of a test program over simple verbal or visual descriptions is that it proves to beginners that an implementation of a project idea in this topic is within reach. They may not understand the program completely, but they see that a few lines of code suffice to implement a certain functionality.

Example Program. An example program illustrates a programming technique and thus combines a topic and programming concepts into a holistic unit. In contrast to a test program, the focus is on the program text – in terms of functionality and style. This Python project illustrates the Python string-method format() and how to use it in a storytelling project.

```
STORY = """ One Saturday morning {she} was walking down
the street with her {thing}. She thought, "How good that
I have the {thing} with me. " """
name = input("Female first name: ")
thing = input("A thing to carry around: ")
text = STORY.format(she=name, thing=thing)
print(text)
```

Example programs can initiate specific *learning* activities like experimenting with the code. Additionally, they can be designed to provoke different *creating* activities, including extending, improving, and transferring.

Extend: The starter project is a very basic program, which must be extended by adding functions. Example: The starter project manages a dictionary and allows nothing more than to look up the translation of a word. Possible extensions are to add new items and to store the updated dictionary.

Improve: The starter program has an extremely low performance. The challenge is to improve the quality of the output and/or the time complexity. E.g., consider a path-finding Python program with a dictionary representing a graph (mapping nodes to lists of adjacent nodes) and a simple recursive function:

```
from random import choice
G = {1: [2, 4], 2: [1, 3, 5], 3: [2, 5],
     4: [1, 5], 5: [4, 2, 3, 6], 6: [5]}

def search_path(node, goal):
    if node == goal:
        return [node]
    else:
        path = []
        while not goal in path:
           path = search_path(choice(G[node), goal)
        return [node] + path
```

The function works fine and returns a path from node to goal, if such exists. But the path may be not the shortest and may even contain loops. This is a sub-optimal result, and the program spends considerable unnecessary time.

Transfer: The starter project can represent functional pattern, which can be applied broadly. The Python program illustrating how to use string methods in a storytelling project can easily be transferred to different projects in a large variety of domains. Another widely applicable pattern is an interactive program, which first reads some data, then processes the data, and finally outputs the results. The creating process does not necessarily imply learning according to the teacher's goals. Sometimes students invest much time for developing a project just by creating beautiful graphics, or adding data without changing the program's structure. For example, in a grade 10 programming class students got a Python starter project, illustrating how to pick a random item from a list. When the user hits the ENTER key, the program prints a "motivating motto" on screen.

```
import random
mottos = ["You look great today!",
          "Never give up.",
          "Yes, you can do it!"]
while True:
    text = random.choice(mottos)
```

```
    print(text)
    input()
```

One group loved this project and developed a version, printing random "fun facts". For several hours, they searched fun facts on the internet and created a very long list with dozens of items. This way they transferred the example program to a new domain and extended it, without learning much about programming.

Architectural Prototype. An architectural prototype is a program that is syntactically correct, but is functionally incomplete. The source text provides the architecture of a more complex project. An example is a graphical user interface with widgets for input and output and buttons that trigger actions when clicked. The function definitions are dummies that have no effect; e.g., in Python syntax:

```
def action():
    pass
```

Another example is a program with incomplete class definitions (attributes and dummy methods) illustrating an aggregate of classes modelling some real structure. An architectural prototype can help to cope with complexity by providing cognitive offload and channeling. When it is the starting point of an iterative development process, the students could start with filling the gaps, for instance by writing function definitions based on verbal specifications (pre- and post-conditions or test requirements). On the one hand, the architectural prototype – since it is already running – offers a test environment. You can start the program, and see from the output whether the self-defined function is working properly. On the other hand, the programming challenge (focusing on mere technical programming concepts) becomes part of a "real" project. Architectural prototypes are sometimes used for written exams because both analytical skills (explaining the program text) and constructive skills (formulating the program text) are tested.

2.2 Starter Projects and Worked Examples

Starter projects are comparable to worked examples in problem centred science education. A worked example is a prototype solution that guides the student through solving the problem, step by step [8]. Worked examples (in science, as well as in mathematics and informatics) are presented in different forms: e.g. in the classroom by a teacher, in explaining videos, in textbooks or animated tutorials [9]. When students solve science problems on their own, they tend to look for work they had recently elaborated in class, and try to adapt these for solving the new task [10]. The advantage of worked examples (compared to problem solving) is the reduction of cognitive load resulting from processes that do not contribute to learning (ineffective load). Like worked examples, starter projects are prototype solutions, and help to reduce cognitive load. They prevent learners from being overwhelmed by the number of information

elements that need to be processed simultaneously, when they start a programming project from scratch. However, one can point out three major differences:

1. A starter project is processed actively and self-controlled, whereas a worked example includes guidance. Instead of following a given path, students explore the code in their own way and change it.
2. A starter project is designed to initiate learning activities and to challenge creativity. It encourages the students to apply certain programming techniques (covering the curriculum) by the given material. But the students can decide, themselves, about design goals. Tinkering with the code may lead to new design ideas, and consequently to research and implement activities that go beyond the scope of the original code of the starter project.
3. A starter project is supposed to be used immediately (and not eventually later like a worked example). It reduces cognitive load (that might act like a barrier) and motivates to start a development and learning process.

These three attributes of starter projects lead to the research questions of a survey, which is presented in the following section.

3 Self-observations on Learning with Starter Projects

This section presents self-observations of university students who have attended an introductory Python course. Two questionnaires were used at two different days covering these research questions:

1. How do students process the code of a starter project?
2. What kind of learning activities do students choose during the development initiated by a starter project?
3. What kind of creative design activities (improving, extending, transferring to a new context) students are interested in?

The University of Münster (Germany) has offered introductory Python programming courses for students of all faculties. The participants can achieve two different types of certificates: everyone who has attended the course regularly receives a certificate of participation without any grade. Students who have successfully passed a written exam at the end of the semester will receive a higher-ranked certificate with a grade. One can assume that at least the exam writers were – additionally to intrinsic motivation – extrinsically motivated to accomplish programming competence during the exercises. The lectures usually have taken place in a computer lab. Each lecture has lasted 180 minutes plus one 30-min-break. Beside presentations and discussions, a main part has been hands-on programming exercises based on different types of starter projects. In each hands-on session, the students could pick from a small repertoire of two or three tasks. In the winter semester 2020/21, this course took place in a digital form (Zoom meetings with breakout sessions for programming exercises). The students were asked to fill questionnaires related to these exercises. One question was whether they intended to take part in the written exam, but no further questions about the person (like age or gender) were asked. Primarily, the questionnaires should encourage

students to reflect on their style of work, which was not seen as a part of a scientific study, but a regular part of the learning process.

3.1 Processing Starter Projects

The first questionnaire was filled by 29 students (group “all”), 12 of them intended to do the written examination (group “exam”) 8 did not (group “no exam”) and the remaining were still undecided. The exercises that day contained 5 tasks with starter projects (type “Example”) the students should try out and change at will. On average, each student worked on 55 % of the tasks. The students were asked to reflect on how they handled starter projects, and to check all statements that applied. Table 1 shows the results.

Table 1. Processing starter projects

Statement	All (N=29)	Exam (N=11)	No Exam (N=8)
1: I typed program text and did not just copy it with drag and drop.	15 (52%)	9 (82%)	2 (25%)
2: I have not copied program text literally but changed it immediately.	11 (38%)	4 (36%)	4 (50%)
3: I have copied program text with drag and drop.	9 (31%)	1 (9%)	4 (50%)
4: I have modified program text in order to explore the effects of instructions	19 (66%)	8 (73%)	7 (88%)
5: Only after I fully understood the program, I tried to implement my own ideas.	10 (34%)	5 (45%)	1 (12%)

It is remarkable that many students (15 out of 29), and especially exam writers (9 out of 11), typed the program texts of the starter projects instead of copying them into the editor with drag and drop. Most students reported that they actively explored code by modifying it.

3.2 Initiated Learning Activities and Effects

Beside tinkering tasks, the exercises on that day included four problem-oriented tasks in the style of exam tasks. In these tasks, the challenge was to write a Python program according to a given verbal specification. Example: “Write an interactive program that calculates the volume of a pyramid.” On average, each student worked on 53% of these tasks. The students were asked to reflect on their work, related to these problem-oriented tasks, and to check statements that applied. The same statements (partly with slightly different wording) were also given for the tinkering tasks with starter projects before. Table 2 compares the results regarding the two types of tasks. The results show that both problem-oriented tasks and tinkering tasks with starter projects initiated a

considerable amount of learning activities, and a feeling of learning success (no significant differences with chi-square statistic on $p < .05$)

Table 2. Initiated learning activities and effects

Statement	Tinkering (N = 29)	Problem (N = 29)
6: I have made experiments in the Python Shell to explore the effects of instructions.	22 (76%)	17 (59%)
7: I felt inspired to develop my own programming project.	9 (31%)	9 (31%)
8: I looked in the course material or other sources for Python commands that I might need.	17 (59%)	21 (72%)
9: I feel like I have learned something new while working on the task.	23 (79%)	24 (83%)
10: I was proud of my result.	6 (21%)	11 (38%)

3.3 Tinkering versus problem solving

The students were asked to make an overall assessment of which type of task they preferred. The majority preferred solving problems over free tinkering.

Table 3. Overall assessment of task types

Statement	All (N=29)	Exam (N=11)	No Exam (N=8)
11: I prefer to tinker with a starter project.	9 (31%)	3 (27%)	2 (25%)
12: I prefer to solve tasks with a problem specification.	15 (52%)	7 (64%)	4 (50%)

3.4 Motivation for Creative Development

On another day, the students got three different types of tasks with starter projects (type: example). They differed in their expectations about what way the given code should be developed. The students were asked to rate their motivation, to extend, improve or transfer starter projects. They had experienced examples of these three types, but they were asked to rate the task type and not the concrete example. The 18 respondents reported to be motivated by all three types of challenges with a median of 4 on a 1-5-Likert scale (1: not motivated, 5: highly motivated).

4 Discussion

4.1 Active writing

A perhaps surprising result of the first questionnaire is that students see writing (typing) rather than copying from the digital hand-out by drag and drop as an appropriate way

to elaborate starter projects. Since they created individual versions of the code by changing and adding words, this can be seen as a special type of "writing to learn" like journal learning [11]. According to [11] the positive effect of journal writing can be increased by appropriate learning strategy prompts. Some successful prompts from journal learning could be adapted to exercises with starter project. An example of an elaborative prompt would be, "Add explaining comments to the program code." A metacognitive monitoring prompt is, "Indicate (by comments) the lines of code, you have problems to understand."

4.2 Desire for challenge

The results of the survey suggest that many students are attracted by the challenge of problem-based tasks (including the risk of failure). To make exercises with starter projects more challenging, teachers could add specific prompts articulating expected goals of development (improve, extend, transfer). Examples for improve-prompts, "Make the program faster.", or "Improve the user interface."

4.3 Creativity

31 % of the respondents of the first questionnaire reported to be inspired to create their own project. In the revised version of Bloom's taxonomy of educational objectives, "Create" is the top category ([12]) in the cognitive process dimension. Accordingly, one might say, that these 31% have high learning ambitions. Constructionism assumes that the wish to create something relevant is a strong motive for learning. However, in this specific context (at the beginning of a programming course), tinkering and exploring code (mainly corresponding to the process categories understand, apply and analyse) seem to be more important than implementing own ideas. According to Csikszentmihalyi, creativity is a social phenomenon and requires a "field" of experts who decide whether an artifact is a novelty within the domain. Learning arrangements with starter projects can get more challenging, if the students are explicitly asked to create something new and unexpected. To make the development a creative process according to Csikszentmihalyi's model [13], students must present their work to the "local field" of classmates who decide about its novelty in some way, for example by rating and commenting. To foster self-efficacy and creative confidence [14], the students should be made aware that there are many ways to add something to a starter project and that at least some of the possibilities are within their reach.

5 Conclusion

Starter projects are digital artifacts that are carefully designed by teachers and textbook authors. In contrast to worked examples (which have a long tradition in math and science education), students and book reader process these code examples in order to learn programming. Starter projects may explain concepts, but they also initiate further exploring activities. Participants of an introductory programming course at a university

report, that they, at least at the beginning, prefer a “learning by writing” approach. Typing code beats copying it by drag and drop. It seems to be important to embed starter projects in an “educational environment” that supports tinkering (by offering additional material and support) and is challenging. Starter projects could be enriched by learning strategy prompts, and creativity challenging prompts.

References

1. Harel, I. E., Papert, S. E.: Constructionism. Ablex Publishing (1991).
2. Resnick, M.: Give P’s a chance: Projects, peers, passion, play. In: Constructionism and creativity: Proceedings of the third international constructionism conference, pp. 13-20. Austrian Computer Society, Vienna (2014).
3. Beck, K.: Extreme Programming Explained Addison-Wesley. Reading, MA (2000).
4. Resnick, M., Rosenbaum, E.: Designing for tinkerability. In: Honey, M., & Kanter, D. E. (eds.): Design, make, play: Growing the next generation of STEM innovators, pp. 163-181, Routledge (2013).
5. Ryoo, J. J., Bulalacao, N., Kekelis, L., McLeod, E., Henriquez, B. (2015, September): Tinkering with “failure”: Equity, learning, and the iterative design process. In: FabLearn 2015 Conference at Stanford University (2015).
6. Mader, A., Dertien, E.: Tinkering as method in academic teaching. In DS 83: Proceedings of the 18th International Conference on Engineering and Product Design Education (E&PDE16), Design Education: Collaboration and Cross-Disciplinarity, Aalborg, Denmark, 8th-9th September 2016 (pp. 240-245).
7. McKay, C., Banks, T., Wallace, S.: Makerspace classrooms: Where technology intersects with problem, project, and place-based design in classroom curriculum. In: International Journal of Designs for Learning, 7(2) (2016)
8. Zhi, R., Price, T. W., Marwan, S., Milliken, A., Barnes, T., Chi, M.: Exploring the impact of worked examples in a novice programming environment. In: Proceedings of the 50th ACM Technical Symposium on Computer Science Education, pp. 98-104 (2019).
9. Van Gog, T., Paas, F., Sweller, J.: Cognitive load theory: Advances in research on worked examples, animations, and cognitive load measurement. In: Educational Psychology Review, 22(4), pp. 375-378 (2010).
10. Chi, M. T., Bassok, M., Lewis, M. W., Reimann, P., Glaser, R.: Self-explanations: How students study and use examples in learning to solve problems. Cognitive Science, 13(2), pp. 145-182 (1989).
11. Nückles, M., Roelle, J., Glogger-Frey, I., Waldeyer, J., Renkl, A.: The selfregulation-view in writing-to-learn: Using journal writing to optimize cognitive load in self-regulated learning. In: Educational Psychology Review, pp. 1-38 (2020).
12. Krathwohl, D. R.: A revision of Bloom’s taxonomy: An overview. Theory into practice, 41(4), pp. 212-218 (2002)
13. Csikszentmihalyi, M.:Creativity. Flow and the Psychology of Discovery and Invention (1996).
14. Jobst, B., Köppen, E., Lindberg, T., Moritz, J., Rhinow, H., Meinel, C.: The faith-factor in design thinking: Creative confidence through education at the design thinking schools

Potsdam and Stanford. In: Design thinking research pp. 35-46. Springer, Berlin, Heidelberg (2012).